

AD-A118 040

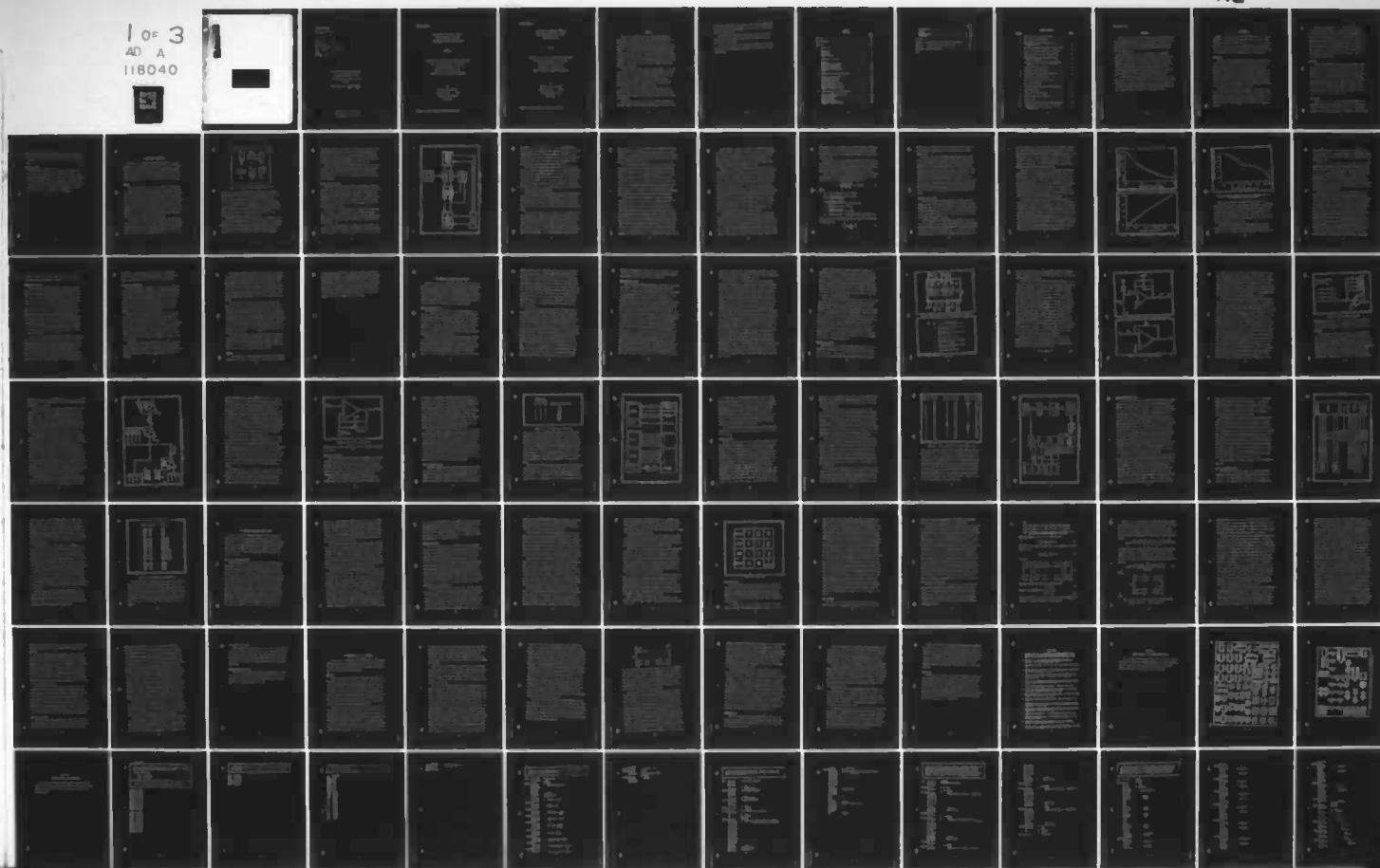
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 13/1
AN AUTOMATED TEMPERATURE CONTROLLER FOR THE ADVANCED HALL EFFEC--ETC(II)

UNCLASSIFIED

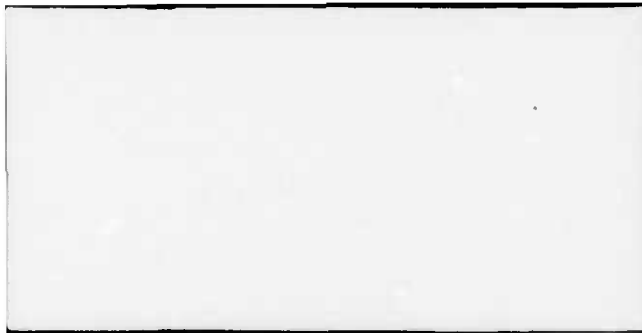
MAR 82 D J PAGE
AFIT/GEO/EE/82M-1

NL

1 of 3
AD A
118040



804 A



AFIT/GEO/EE/82M-1

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



AN AUTOMATED TEMPERATURE CONTROLLER
FOR THE ADVANCED HALL EFFECT
EXPERIMENTAL DATA ACQUISITION SYSTEM
THESIS

AFIT/GEO/EE/82M-1 Daniel J. Page
2Lt USAF

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

AFIT/GEO/EE/82M-1

AN AUTOMATED TEMPERATURE CONTROLLER
FOR THE ADVANCED HALL EFFECT
EXPERIMENTAL DATA ACQUISITION SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

by

Daniel J. Page, B.S.E.E.

Second Lieutenant USAF

Graduate Electro Optics

March 1982

Approved for Public Release; Distribution Unlimited.

AFIT/GEO/EE/82M-1

AN AUTOMATED TEMPERATURE CONTROLLER
FOR THE ADVANCED HALL EFFECT DATA
ACQUISITION SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

by

Daniel J. Page, B.S.E.E.

Second Lieutenant USAF

Graduate Electro Optics

March 1982

Approved for public release; distribution unlimited.

Preface

Dr. Patrick M. Hemenger of the Air Force Wright Aeronautical Laboratories, Materials Laboratory (AFWAL/ML), Wright-Patterson Air Force Base, Ohio, developed and built a Hall effect experiment to gather data on the impurity levels and electrical properties of silicon samples. In 1979, this experiment was totally automated with the exception of temperature control. Starting with the experimental equipment and the automation system, I designed and implemented an automatic temperature controller which would have completed the automation of the experiment. This report details the process of designing and implementing this controller.

I want to thank Maj. Alan A. Ross, my thesis advisor, for his help and guidance. His advice was invaluable during the hardware and software development. In addition, he went over multiple drafts of this report in detail to help shape it into a more readable product. I would also like to thank the other members of my thesis committee: Dr. Thomas C. Hartrum and Dr. Theodore E. Luke.

I also want to thank Dr. Hemenger, the thesis sponsor who provided the thesis topic. He went out of his way to insure that I had all the necessary information and equipment to complete the design and implementation.

Special mention must go to Mr. Ronald E. Perrin of AFWAL/ML and Mr. Dane C. Hanby of the University of Dayton Research Institute. Both

of these men provided their skill and expertise during the implementation of the hardware and software designs. Without their help, the controller could not have been built.

I would also like to thank Mrs. Diane Katterheinrich who proofed and typed the final copy of this report and to Lt. David A. Huss who drafted most of the figures.

Finally and most importantly, I would like to thank my wife, Laura. Without her love, understanding, and support this thesis would not have been finished.

Contents

	<u>Page</u>
Preface	ii
List of Figures	vi
Abstract	vii
I. Introduction	1
Background	1
Problem	2
Constraints	2
Scope of Thesis	3
II. Requirements Definition	4
AHEEDAS	4
Experimental Environment	6
Temperature Measurement	12
Temperature Adjustment	17
User Imposed Requirements	18
Summary	19
III. Hardware Design and Implementation	21
Hardware Design Overview	21
Microprocessor System	23
Thermometer Interface	25
Heater Coil Interface	34
LSI-11 Computer Interface	35
Display Interface	38
Switch and Keypad Interface	42
Summary of Hardware Design	43
IV. Software Design and Implementation	47
Use of the MC6809 Architecture	47
Supporting Software	49
Control Scenario	52
Control Scheme	53
Required Information	59
Summary of Software Design	61

	Page
V. Recommendations and Conclusion	62
Results	62
Recommendations	66
Conclusion	68
References	69
Appendix A: Controller Board Layout and Pin Connections . . .	A-1
Appendix B: Controller Program Assembly Listing	B-1
Appendix C: Operator's Manual	C-1
Vita	

List of Figures

<u>Figure</u>		<u>Page</u>
1	van der Pauw Sample Configurations	5
2	Block Diagram of the AHEEDAS	7
3	Carbon-Glass Thermometer Resistance vs. Temperature	14
4	Platinum Thermometer Resistance vs. Temperature . . .	14
5	Silicon Thermometer Voltage vs. Temperature	15
6	MC6821 Peripheral Interface Adapter (PIA)	26
7	MC6821 Peripheral Interface Adapter (PIA) Register Selection	26
8	Control Thermometer Constant Voltage Biasing Source	28
9	Classical Inverting Amplifier	28
10	Modified Inverting Amplifier	30
11	Thermometer Interface Detection Section	32
12	A/D Reference Voltage Circuit	34
13	HP 6130C Current Limit Data Format	36
14	Heater Coil Interface	37
15	Automatic Temperature Controller Character Set	40
16	Display Interface	41
17	Switch and Keypad Interface	44
18	Microprocessor System Memory Map	46
19	Keypad Layout	52
A-1	Microcomputer Board Layout and Pin Connections	A-2
A-2	Thermometer Interface Board Layout and Pin Connections	A-3

Abstract

The Air Force Wright Aeronautical Laboratories, Materials Laboratory, conducts experiments using the Hall effect to characterize the electrical properties and impurity levels of silicon samples. An automated data acquisition system controls the conduct of the experiment and reduces all the necessary data.

The purpose of this study was the development of an automated temperature controller to interface with the automated data acquisition system and the experiment. The temperature controller is designed to control the temperature of the silicon sample to within 0.005 degrees Kelvin in the temperature range of 4.2 to 300 degrees Kelvin. The control algorithm measures the thermal impulse response of the system and uses this information to adjust and control the temperature.

An MC6809 microprocessor with 10K bytes of EPROM and 640 bytes of RAM is used to implement the controller. The control algorithm and other software was developed to enable the controller to control temperature. A number of problems with the present controller design are identified and recommendations for improvements to the design are made.

I. Introduction

The Air Force Wright Aeronautical Laboratories, Materials Laboratory (AFWAL/ML), Wright-Patterson Air Force Base, Ohio, uses a Hall effect experiment to analyze the dopant and impurity content of silicon samples. They wanted to develop an automatic temperature controller in order to complete the total automation of their experiment. This report describes the requirements, design, and implementation of this controller.

Background

AFWAL/ML is tasked with determining the impurity concentrations in various semiconductor samples used for optical detectors and for very high speed integrated circuits. The concentrations and activation energies of electrically active impurities are important quantities for the complete characterization of these semiconductors. Measurements of sample resistivity and Hall voltage versus temperature enable the experimenter to determine the electrical transport properties of a semiconductor sample. From these measurements, the net impurity concentrations and carrier mobility can be determined. [Ref. 1]

The experiment was originally done manually. It took from six to eight hours to make all the necessary measurements on one sample. Using the van der Pauw and the classical Hall bar methods, 600 to 1200 voltage versus current measurements were made over a temperature range of 4.2 to 300 degrees Kelvin. While successful, this experiment was awkward to perform.

In 1979, Captain Edgar A. Verchot, Jr. automated the control of the experiment and data acquisition using an LSI-11 microcomputer system. The Advanced Hall Effect Data Acquisition System (AHEEDAS), as it is called, controls all aspects of the experiment except for the temperature control. Temperature control remained manual at the user's request. Temperature control under AHEEDAS, as under the manual experiment, was accomplished using an analog controller. Using AHEEDAS lessened considerably the amount of time and effort needed to collect data, however, much valuable technician time was lost in monitoring the analog temperature controller.

Problem

To relieve the technicians from monitoring the experimental apparatus, the user wanted to automate the temperature controller. One requirement for this controller was that it be compatible with the AHEEDAS system. In addition, the user wanted to have the capability to use this same automatic temperature controller for future projects requiring temperature control. Hence, temperature control could not be implemented using the AHEEDAS computer system. The purpose of this study, therefore, was to design and implement a stand-alone automatic temperature controller that is compatible with the present experimental set-up and can be used for future projects.

Constraints

During the development of AHEEDAS, a Hewlett-Packard Model 6130C Digital Voltage Source was purchased by the user. In order to reduce the cost of this project, it was requested that this source be used, if possible, to drive the heating element. In addition to this constraint,

the user also requested that the temperature controller have the capability of being used not only in an automatic mode under the control of the LSI-11, but also in a manual mode under the control of the technician.

Scope of Thesis

A discussion of the experimental environment and AHEEDAS is contained in Chapter II. The requirements for the automatic temperature controller based upon this discussion are also contained in this chapter. Chapter III details the hardware design and implementation of the controller. The control scheme and supporting software are the main subjects covered in Chapter IV. Finally, recommendation for improvements to the automatic temperature controller are discussed in Chapter V.

II. Requirements Definition

Before detailing the hardware design and implementation, the requirements on the temperature controller must be defined. This chapter contains discussions on the AHEEDAS system, experimental environment, temperature measurement, and temperature adjustment which define some of these requirements. The remaining requirements are based upon the user's desires and constraints and are also included in this chapter.

AHEEDAS

Before the requirements on the automatic temperature controller were defined, the Hall experiment and the AHEEDAS system were examined. As was stated in Chapter I, the purpose of this experiment is to measure the impurity concentrations in semiconductor samples. To accomplish this, two methods of data acquisition can be used in the Hall experiment: the van der Pauw method and the classical Hall bar method. These two methods differ only in the configuration of the sample contacts and the equations used to derive the parameters of interest: resistivity, Hall mobility, carrier concentration, and the Hall coefficient. Therefore, only the van der Pauw method will be discussed here.

The van der Pauw technique uses a sample having only four electrical contacts (Fig. 1). In each of the six configurations, the potential difference between V1 and V2 is recorded. Additionally, these measurements are repeated with the biasing voltage reversed. In sample configurations (e) and (f), the voltage difference between V1 and V2 with a

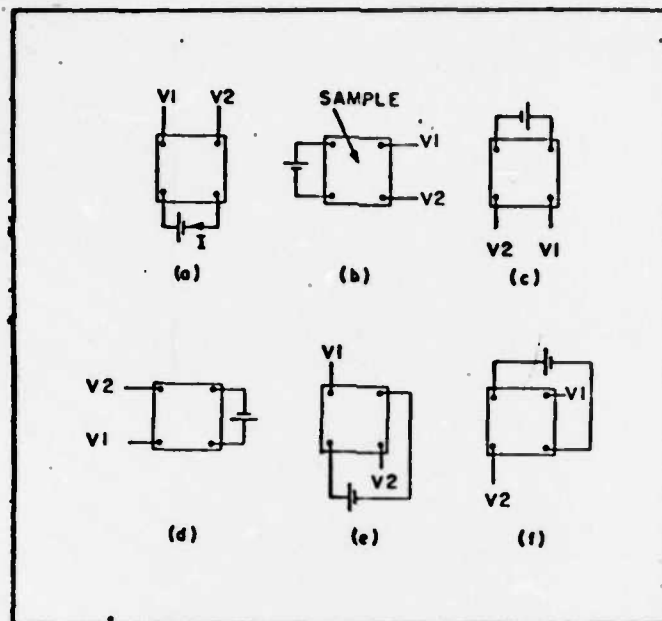


Figure 1. van der Pauw Sample Configurations [Ref. 1]

magnetic field applied perpendicularly to the sample is also measured. Both polarities of the field must be used for these measurements. From these data, the four parameters of interest are calculated. The van der Pauw technique is then repeated at a full range of temperatures from 4.2 to 300 degrees Kelvin yielding sample resistivity, Hall mobility, carrier concentration, and the Hall coefficient as a function of temperature. Further analysis of these data produces information on the impurity concentration in the sample. [Ref. 1]

The AHEEDAS system controls all aspects of the Hall experiment with the exception of temperature control. This system controls the switching between each of the various sample configurations, measures and controls the magnetic field strength, takes all necessary sample measurements, and reduces the sample measurements to obtain the output parameters.

The AHEEDAS produces a list of sample resistivity, Hall mobility, carrier concentration, and the Hall coefficient as a function of temperature. This output is stored in data files on floppy disk storage along with all of the raw data from the experiment. The output is also printed on an interactive terminal as the experiment is done. [Ref. 1]

The only remaining gap in the AHEEDAS system is temperature control. Because of this, the AHEEDAS system cannot conduct the Hall experiment unattended. An automated temperature controller that interacts with both the AHEEDAS system and the Hall experiment, however, would alleviate this problem.

Experimental Environment

Before any serious work could be done on the automatic temperature controller, the experimental environment had to be understood. To determine these requirements, three areas were examined: the physical arrangement of the major components of the experiment, the method of data acquisition, and the precision of the final results of the experiment. These examinations formed the basis upon which further analysis of the requirements could be based.

In examining the physical arrangement of the experimental apparatus, six major components were identified. These included: the sample holder, the cooling hardware, the electromagnet, the biasing sources and instrumentation, the LSI-11 computer system, and the analog temperature controller (Fig. 2).

The first of these components, the sample holder, essentially consists of a copper bar upon which the silicon sample is placed. Attached to this copper bar is a 50 ohm coil of copper wire which is used to heat

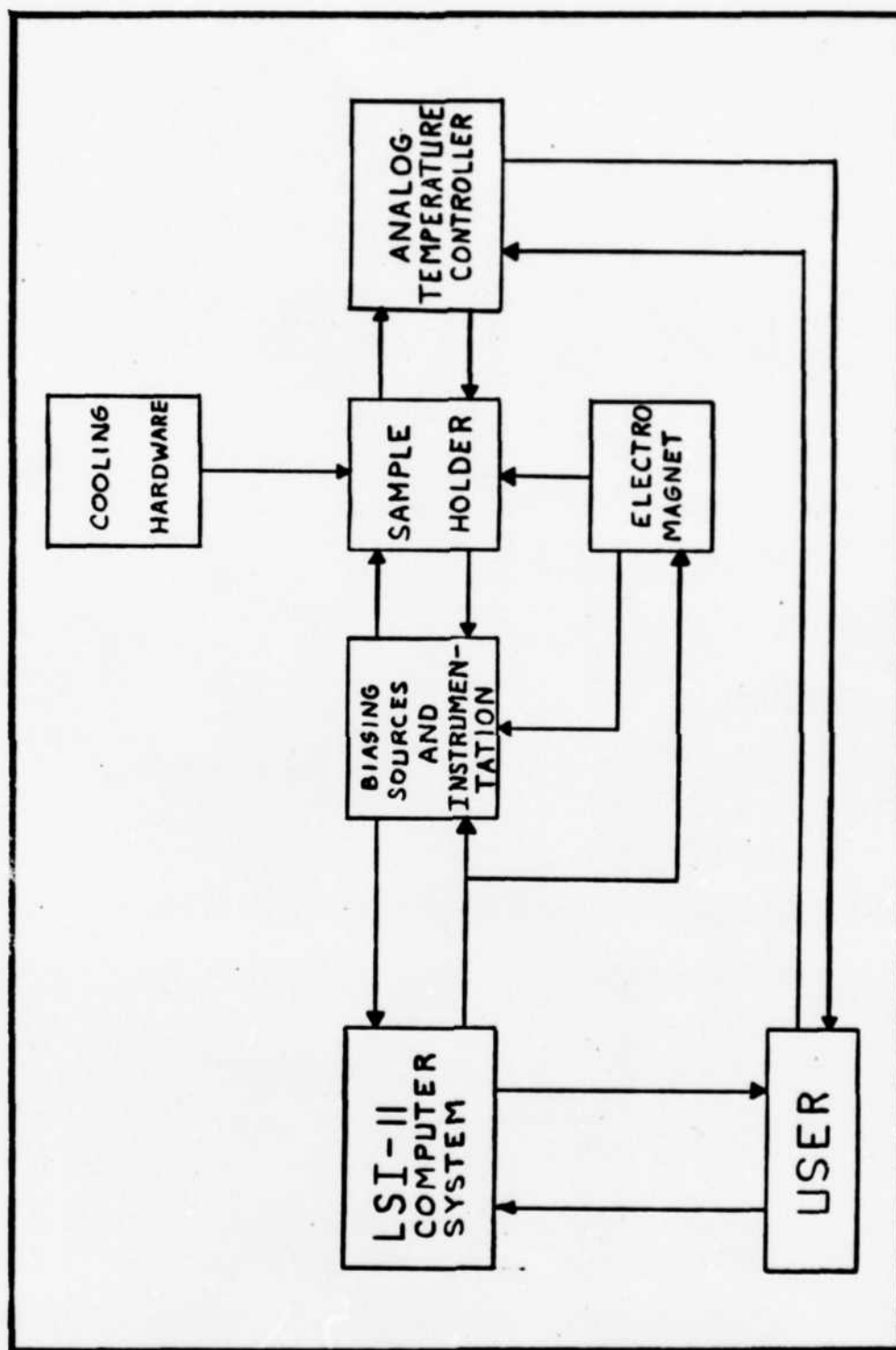


Figure 2. Block Diagram of the AHEADAS

the sample. Directly under the bar are located three thermometers: a silicon diode, a carbon-glass resistor, and a platinum resistor. These thermometers are used for temperature measurement and control. The silicon sample and the three thermometers are electrically connected to coaxial leads which carry the signals to and from the sample holder. The entire sample holder is covered with a copper cover.

The second of the experimental components is the cooling hardware. This is an open loop cooling system consisting of a liquid helium source, a diffuser, and a triple-walled dewar. The liquid helium flows through the diffuser and is vaporized as it enters the dewar. The sample holder is placed in this helium stream. The exhaust gas is then carried out the top of the dewar. With this system, it is possible to obtain temperatures of 4.2 degrees Kelvin, the boiling point of helium, with little or no effect from the room environment.

The third component of the experimental system is the electromagnet. Both the van der Pauw and the Hall bar measurement techniques require that large magnetic fields be placed perpendicular to the sample. For this reason, the dewar is positioned between the poles of the electromagnet in such a way as to place the sample perpendicular to the magnetic field.

The fourth major component of the system is the instrumentation and the biasing sources. Included in this component of the system is all the metering needed to measure the voltage, current, and magnetic field strength. Also included in this component are the sources needed to place the proper bias on the silicon sample and the switching network used to change to each of the various configurations of the van der Pauw and Hall bar measurements. All of this equipment is compatible with a

digital interface. Of particular interest to the design of the temperature controller is the equipment used to measure temperature. This consists of a 10 microampere constant current source to bias the silicon diode thermometer and a Hewlett-Packard Model 3455A Digital Voltmeter to measure the voltage across this diode.

The fifth component of the system is the LSI-11 computer system. This system controls the actual execution of the experiment and was therefore very important in the design of the temperature controller. This system consists of a Digital Equipment Corporation LSI-11 micro-computer system with 28K words of memory, a floppy disk system, an interactive terminal, and an acoustic coupler. The LSI-11 interfaces with some of the experimental equipment through DRV-11 parallel interface cards. The remaining equipment is interfaced through an IEEE 488-1975 standard bus. Contained in this computer system is the AHEEDAS program and a calibration table for the silicon diode thermometer.

The final component of the experimental system was the analog temperature controller. This controller, the Artronix Model 5301, was connected to the sample heater and the two control thermometers, the carbon-glass and platinum resistors. These connections completed a closed-loop temperature control system between the Artronix controller and the sample holder. The appropriate control thermometer was switch selectable from the front of the analog controller. Using this analog controller with the two control thermometers enabled temperature to be controlled to within approximately 0.01 degrees Kelvin. The Artronix analog controller supplied the sample heater with an output voltage of 0 to 50 volts with a maximum current of 1 ampere. This output corresponded to a temperature range of 4.2 to approximately 300 degrees Kelvin.

After examining the physical arrangement of the experimental apparatus, the next area that needed to be studied was that of data acquisition. Of particular interest in this study was not how the data were obtained from the silicon sample, but rather the steps involved in reaching a temperature set point and the events that occurred during data acquisition that affected temperature control.

Originally, a typical data acquisition cycle began with the LSI-11 displaying a prompt on the terminal. This prompt specified a temperature set point and the corresponding voltage for the silicon diode and asked the user to adjust the analog controller until this specified voltage is reached. (The silicon diode voltage/temperature calibration table is contained in the LSI-11 computer.) At this point, the user decided which control thermometer to use. Typically the carbon-glass resistor is used at temperatures below 60 degrees Kelvin and the platinum resistor is used above this point. After the appropriate choice of thermometer was made, the user set the requested voltage by adjusting the analog controller while monitoring the diode voltage on the digital voltmeter. Once the user determined that the voltage was properly set, he then signaled the LSI-11 by typing a 'GO' command on the terminal. The LSI-11 then began the actual data acquisition process.

During some points in the data acquisition process, the magnetic field is raised to 1 kilogauss. The LSI-11 needs precise temperature measurements. However, due to the high magnetoresistance of the silicon diode, these temperature measurements can not be made while the field is on. (It is for this same reason that the silicon diode can not be used for temperature control.) To alleviate this problem, the temperature is measured with the silicon diode just before the field is turned on with

the assumption that the temperature is controlled adequately during the on-field times. After all data are collected, the magnetic field is returned to zero and the LSI-11 displays the prompt for the next temperature set point.

Once the data acquisition process was understood, the next area to be investigated was the precision of the final results of the experiment. In this case, the main concern was how precise the temperature had to be maintained in order to have the desired precision in the results.

Carrier concentration is the most sensitive quantity to variations in temperature. From solid state physics theory, the number of holes per unit volume in a p-type semiconductor is given by:

$$p = 2 \left[\frac{2\pi m_h^* kT}{h^2} \right]^{3/2} \text{EXP}(-E/kT) \quad (1)$$

where

p = hole concentration (#/cubic meter)

m_h^* = reduced mass of a hole in silicon
= $5.37\text{E-}31$ kilograms

k = Boltzmann's constant
= $8.62\text{E-}5$ electron volts per degree Kelvin

T = temperature (degrees Kelvin)

h = Planck's constant
= $4.14\text{E-}15$ electron volt-seconds

E = ionization energy of the dopant (electron volts) [Ref. 2]

From equation (1) it was determined that the relative error in temperature is given by:

$$\frac{dT}{T} = \frac{1}{3 + \frac{2E}{kT}} \cdot \frac{dp}{p} \quad (2)$$

Since the effect of temperature variation is inversely proportional to the temperature, temperature control is more critical at the lower temperatures.

A typical sample used in this experiment is boron-doped silicon which has an ionization energy of 0.045 electron volts. Measurements are usually taken down to a temperature of 18 degrees Kelvin for this substance, due to the increasing resistivity of the sample with decreasing temperature. The user wanted to be able to measure the carrier concentration with a relative error on the order of one percent. Upon substitution these values into equation (2), it was found that the temperature had to be controlled to within less than 0.01 degrees Kelvin. Similar values for temperature deviation were obtained for other dopant materials. Based upon this information, the required temperature deviation was assumed to be 0.005 degrees Kelvin for all further calculations.

Temperature Measurement

Once an understanding of the experimental environment had been gained, a closer look at temperature measurement was needed. In this examination, three questions were answered: Why were three thermometers needed, how did these three thermometers measure temperature, and what requirements would be implied if these three thermometers were used in the automatic temperature controller?

In the discussion on the experimental environment, it was shown that three thermometers are located in the sample holder: a temperature measurement thermometer (the silicon diode) and two control thermometers (the carbon-glass and platinum resistors). It was also mentioned that the silicon diode has a high magnetoresistance and therefore can not be

used for temperature control or measurement while the magnetic field is on. Both the carbon-glass and the platinum thermometers have a negligible magnetoresistance making them ideal for temperature control. These thermometers, however, lack the repeatability which the silicon diode possesses. Neither of them, therefore, were adequate for accurate temperature measurement. For this reason, separate thermometers are needed for temperature measurement and control.

The justification for using two control thermometers was shown by examining the plots of resistance versus temperature for the two thermometers and recalling that temperature control was more critical at lower temperatures (Figs. 3 & 4). Clearly, the carbon-glass resistor met this criteria however, it does not have the sensitivity to operate above approximately 60 degrees Kelvin. Therefore, a second control thermometer was needed -- the platinum resistor. The combination of these two thermometers enabled the analog controller to monitor variations in temperature over the entire temperature range.

To answer the question of how the three thermometers measured temperature, it was necessary to examine the two control thermometers separately from the silicon diode thermometer. The control thermometers, as stated in the previous section, interfaced only with the analog controller. This controller provided a constant voltage to these two thermometers and measured the current flowing through them. In this way, the controller could measure the resistance of the two thermometers, and thus the temperature. The silicon diode, on the other hand, is forward biased with a 10 microampere constant current. This current yields a voltage across the diode which varies with temperature (Fig. 5).

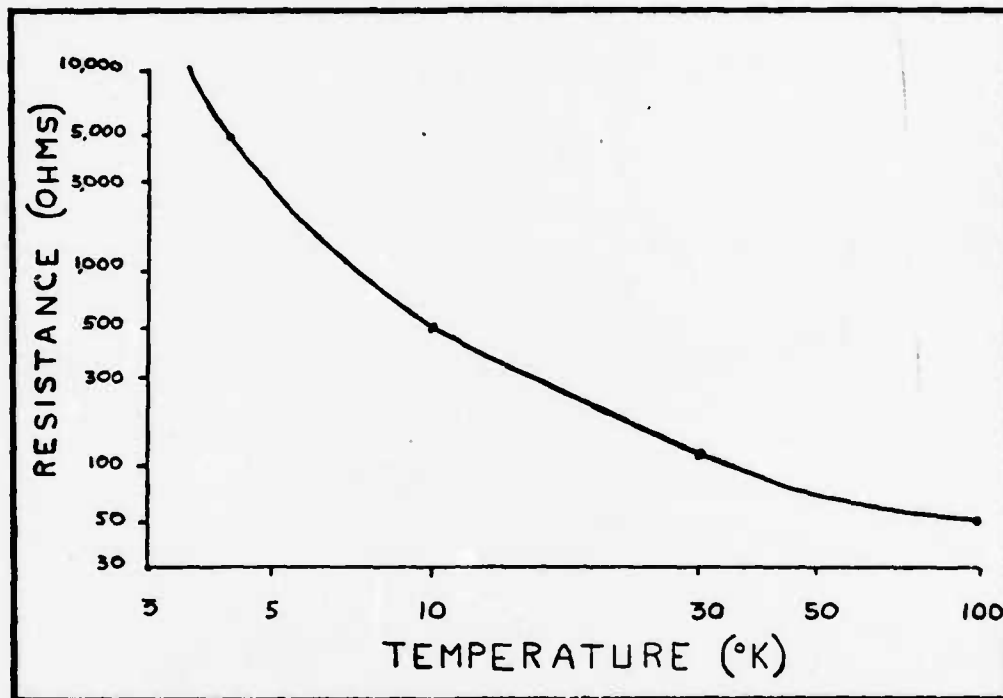


Figure 3. Carbon-Glass Thermometer Resistance vs. Temperature [Ref. 3]

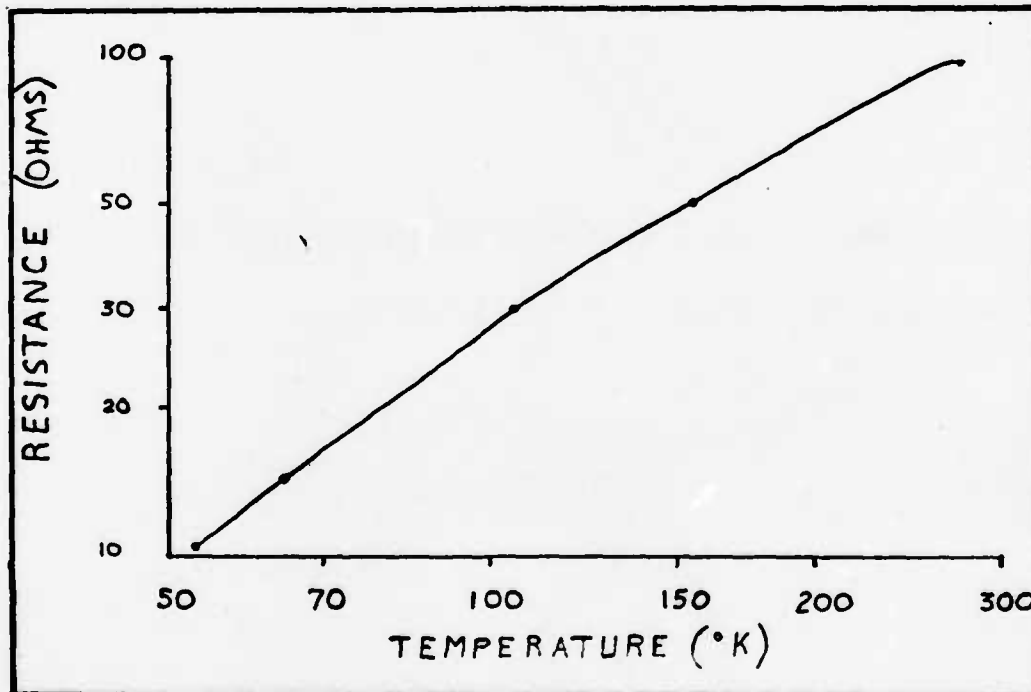


Figure 4. Platinum Thermometer Resistance vs. Temperature [Ref. 4]

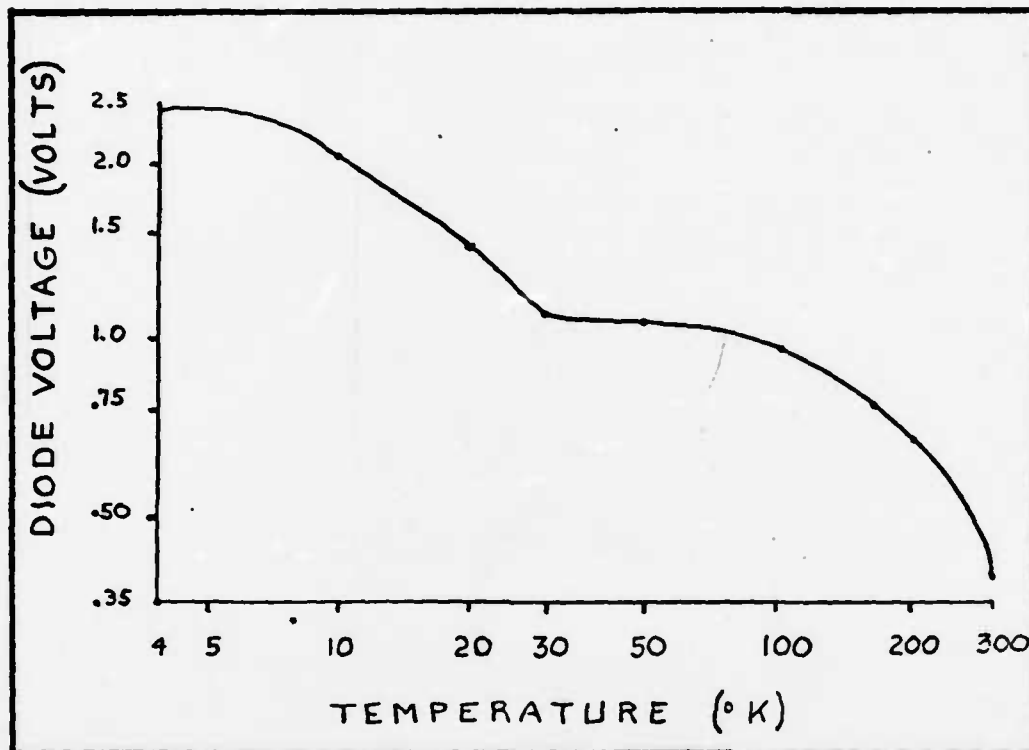


Figure 5. Silicon Thermometer Voltage vs. Temperature [Ref. 5]
(10 microampere forward bias current)

Based upon the rationale for using three thermometers and the knowledge of how these thermometers measured temperature, it was possible to place some additional requirements on the automatic temperature controller. Namely, the functional blocks required to interface the controller with the thermometers were defined. In addition, the degree of precision to which the controller has to measure the input signals from the three thermometers was determined.

In order to have a completely automated temperature controller (one in which the temperature could be changed accurately from one temperature level to another), the silicon diode has to be monitored by

the automated controller. This is because the silicon diode is the only calibrated measure of the actual temperature of the sample. Therefore, this controller has to be able to measure the voltage signal from the silicon diode.

The automated controller also has to interact with the two control thermometers. In addition, both of these thermometers need a biasing source, since the original biasing source, the analog controller, was to be replaced by the automated controller. Two schemes were possible for biasing the thermometers and measuring the resulting signals: placing a constant voltage across the thermometers and measuring the resulting currents or placing a constant current through the thermometers and measuring the resulting voltages. The actual scheme used was determined during the hardware design.

The requirements on the signal measurement sensitivity for the automatic temperature controller were determined from examining the plots of voltage versus temperature for the silicon diode and resistance versus temperature for the two control thermometers. Using a temperature deviation of 0.005 degrees Kelvin, it was found that the silicon diode voltage has to be measured to within one part in 7,000 at 4.2 degrees Kelvin and to within one part in 30,000 at 300 degrees Kelvin. The measurement sensitivity for the two control thermometers was evaluated at 60 degrees Kelvin, the changeover point between the two thermometers. This point was also the point of least sensitivity for these two thermometers. Again, using a temperature deviation of 0.005 degrees Kelvin, the required sensitivity was found to be one part in 18,000 for the carbon-glass resistor and one part in 7,000 for the platinum resistor. The signal measurement equipment of the automatic

temperature controller is required, therefore, to measure the current and voltage signals to within these tolerances.

Temperature Adjustment

After examining the thermometers and the temperature measurement scheme, a clearer understanding of the temperature measurement process was needed. Temperature adjustment was originally accomplished by the user and the Artronix analog temperature controller. The user, while monitoring the silicon diode thermometer voltage, made adjustments to the analog controller until the desired voltage was obtained. The analog controller, in turn, controlled the power being sent to the heater coil in the sample holder. Once the proper diode voltage was obtained, the analog controller could maintain the temperature by monitoring the input from the selected control thermometer and making the appropriate adjustments to the heater coil power. The automated temperature controller has to perform all of these tasks: monitoring the silicon diode, adjusting the power to the heater coil, selecting and monitoring the proper control thermometer, and changing or maintaining a temperature.

Some special quirks of the system had to be considered in performing these tasks. It was noted when changing from one temperature to another using the analog controller that the system reacted quite differently at different temperature ranges. At temperatures below approximately 40 degrees Kelvin, the thermometers reacted almost instantaneously to changes in the heater coil power. However, at temperatures above approximately 150 degrees Kelvin, the thermometers took from one to three seconds to 'see' heater coil power changes. This was accounted for by

the change in the thermalconductivity of the copper sample holder [Ref. 6]. The automatic temperature controller has to account for this change in thermalconductivity in its control of the temperature.

User Imposed Requirements

Besides the requirements imposed on the automatic temperature controller by the experimental environment and the temperature measurement and adjustment techniques, certain user requirements were also imposed. The user wanted the automatic temperature controller: (1) to be compatible with the existing experimental equipment, (2) to have an optional manual mode of operation not under the control of the LSI-11 computer system, and (3) to be usable on similar projects needing temperature control that may occur in the future.

The first of these requirements, compatibility with the existing experimental equipment, dictates that the automatic temperature controller be able to communicate with the LSI-11 computer system, the thermometers, and the heater coil in the sample holder. In anticipation of an automatic temperature controller, one DRV-11 parallel interface card was reserved on the LSI-11 for use by this controller. It was, therefore, required that the automatic temperature controller be able to interface with the DRV-11 in order to establish a communication link with the LSI-11.

The thermometer interfacing requirements were defined in the previous sections. However, compatibility with the existing system did impose an additional requirement. The automatic temperature controller could not interfere with the data acquisition process of the LSI-11 computer.

The automatic temperature controller is also required to interface with the heater coil. Again, in anticipation of the automatic temperature controller, a Hewlett-Packard Model 6130C Digital Voltage Source was procured by the Materials Laboratory. This voltage source, like the Artronix analog controller, can supply the heater coil with a 0 to 50 volt output at 1 ampere. However, this voltage source has a digital interface compatible with microcircuit logic levels. Therefore, the automatic temperature controller was required to interface with the HP 6130C.

The second requirement was to have an optional manual mode of operation not under the control of the LSI-11 computer system. This required that the automatic temperature controller be able to communicate with the user independent of the LSI-11. To meet this requirement, it was established that the automatic temperature controller would need an alphanumeric display so that the controller could prompt the user and so that the user could monitor the status of the controller. In addition, the controller needed an input medium, a keypad, so that the user could provide the information to the controller that would normally be provided by the LSI-11.

The third user imposed requirement was for the automatic temperature controller to be usable on future projects requiring temperature control. This requirement established that the automatic temperature controller would need to be easily modifiable. Therefore, it was decided that the controller would be microprocessor based.

Summary

This chapter defined the requirements on the automatic temperature controller. Based upon the discussions on the experimental environment,

on the temperature measurement and adjustment techniques, and on the user imposed requirements, it was established that the automatic temperature controller would be a microprocessor based system. It would, at a minimum, have to interface with the user, the LSI-11 computer system, all three of the thermometers, and the heater coil. Based upon these requirements, it was possible to begin the design of the hardware and of the software needed to make the automatic temperature controller a reality. The next two chapters describe this design.

III. Hardware Design and Implementation

Once the requirements of the automatic temperature controller were defined, it was possible to begin the hardware design and implementation. This chapter describes the design of each major component of the hardware: the microprocessor system, the thermometer interface, the heater interface, the LSI-11 computer interface, the display interface, and the switch and keypad interface. Each of the design decisions made were related to the requirements defined in the previous chapter.

Hardware Design Overview

Before designing the hardware for the automatic temperature controller, a philosophy for the design was established. Many important considerations needed to be addressed. However, there was no basis in the requirements to cover these areas. Two questions, in particular, were raised at this time: should the controller be built from microcomputer boards available on the market or should it be built from scratch, and what would be the time required for the microprocessor to perform all the operations required of it.

The question of building the controller from microcomputer boards or from scratch generated many thoughts. While the cost of the pre-built computer boards was higher than what a system built from scratch would cost, the pre-built boards would eliminate the time required for board layout, bus timing considerations, and some of the software writing. One of the main disadvantages, however, was that the electronic

hardware needed to build a temperature controller came on boards that also contained hardware not needed. Therefore, a significant part of the final cost of the temperature controller would be for electronics that would never be used. The scratch system, on the other hand, gave the freedom to configure the controller with only the electronics that were absolutely needed and with a choice of any electronic chips available. Since the automatic temperature controller would be, at this time, an untried design, there was a possibility that the hardware acquired would not do the job. With this in mind, it was hard to justify the expense of the pre-built computer boards. Therefore, it was decided that the hardware for the automatic temperature controller would be built from scratch.

The second question asked how much time the microprocessor could spend performing all its required operations. This question, however, was impossible to answer at this time, since the required operations, the microprocessor, the clocking rate, and other variables were not yet defined. But this was a question that needed to be addressed. The answer to this question would determine whether housekeeping chores such as display updating and switch contact debouncing would be done in software or in hardware. It would also determine how fast the microprocessor and memory had to be. Since there was no reasonable answer to this question, it was decided that all housekeeping operations that could easily be implemented in hardware would not be implemented in the software. In this way, the microprocessor would have some extra time, if needed, to perform its primary function -- control temperature. A decision was also made about the speed of the microprocessor and memory. The slower speed chips were to be used where a choice was available.

However, the design was also to be compatible with the faster devices. In this way, substitutions could quickly be made if during testing it was determined that the faster devices were required.

Microprocessor System

After determining the design philosophy, the microprocessor system was designed. This system involved three components: the microprocessor, the memory, and the interfaces. The first step in the design of the microprocessor system was choosing a microprocessor. Due to a greater familiarity with Motorola products, it was decided that a Motorola microprocessor would be used. Of the microprocessor and micro-computer chips produced by Motorola, the MC6809 microprocessor seemed to be best suited for use on the automatic temperature controller. This microprocessor has many features that made it look attractive for this design: an explicit multiply instruction, two stack pointers, and two index pointers. In addition, the MC6809 has some 16-bit capability that enables it to add, subtract, load, and store a 16-bit operand. Another feature of the MC6809 is its enhanced indexed addressing capability. This enables this microprocessor to do indexed and indirect addressing from all of the pointer registers: the stack pointers, the index pointers, and the program counter. While all of these features were appreciated during the software development, the MC6809 also has features that were important during the hardware design. The MC6809 has an on-chip clock which produces all the clocking signals needed for the microprocessor and the control bus. Therefore, the only external clocking hardware needed was a crystal to control the speed of the on-chip clock. An additional hardware feature of the MC6809 microprocessor

is the three interrupt lines: the non-maskable interrupt, the normal interrupt, and the fast interrupt line. The uses of each of these lines will be described later in this report. [Ref. 7]

The next component of the microprocessor system is the memory. Two forms of memory were used: Read Only Memory (ROM) for the program to reside in, and Random Access Memory (RAM) for the data to reside in. Since it was not known how much memory would be needed, an arbitrary choice of 1K bytes of RAM and 4K bytes of ROM was made. During the software development, however, it was discovered that the 4K bytes of ROM were too small to hold the controller program. In order to allow enough memory for the program, 10K bytes of ROM was required. Unfortunately, the hardware was already being built during the software development and the number of sockets available was limited. Therefore, it was necessary to use some of the RAM sockets to increase the size of ROM. The final amount of memory used was 10K bytes of ROM and 640 bytes of RAM. Once the amount of memory to be used was known, the specific chips needed to implement this memory requirement were determined. The RAM chip selected was the Motorola MCM6810 RAM. This was a common device and relatively cheap. The MCM6810 contains 128 bytes and, therefore, 5 of these devices were needed to implement the requirement of 640 bytes of RAM. Since it was anticipated that many changes would have to be made to the controller program, an Erasable Programmable ROM (EPROM) was selected for the program to reside in. The chip selected was the MCM2716. Again, this was a common chip and could be obtained from many different manufacturers. Each 2716 contains 2K bytes. Therefore, the total number of devices, both RAM and EPROM chips, needed was ten. [Ref. 7]

Once the microprocessor and memory chips were established, the remaining component of the microprocessor system, the interfaces, had to be established. To make both the hardware and the software designs simpler, all the communication between the microprocessor system and the outside world was to be done through Peripheral Interface Adapters (PIA). A standard Motorola compatible PIA is the MC6821. This device features two 8-bit parallel ports and four control lines on the peripheral side (Fig. 6). A detailed description of the MC6821 PIA and its operation is contained in Reference 8.

Some atypical design decisions were made in regard to the addressing of the PIA. As shown in Figure 7, two register select lines (RS1 and RS0) are needed to address the six internal registers of the MC6821. Typically, these lines are connected to the two least significant address lines of the microprocessor (A1 and A0 respectively). These connections place control register A between the A and B peripheral data registers. In order to make full use of the 16-bit load and store capability of the MC6809 microprocessor, however, it was desirable to locate the two peripheral data registers in adjacent memory locations. This was done by specifying that RS1 should be connected to the least significant address line (A0) and that RS0 should be connected to one of the other address lines. A further discussion of the PIA addressing is contained in the summary of the hardware design.

Thermometer Interface

Once the microprocessor system was established, each of the interfaces had to be designed. The first of these interfaces is the thermometer interface. In order to meet the requirements stated in Chapter II, this interface has to enable the microprocessor system to measure

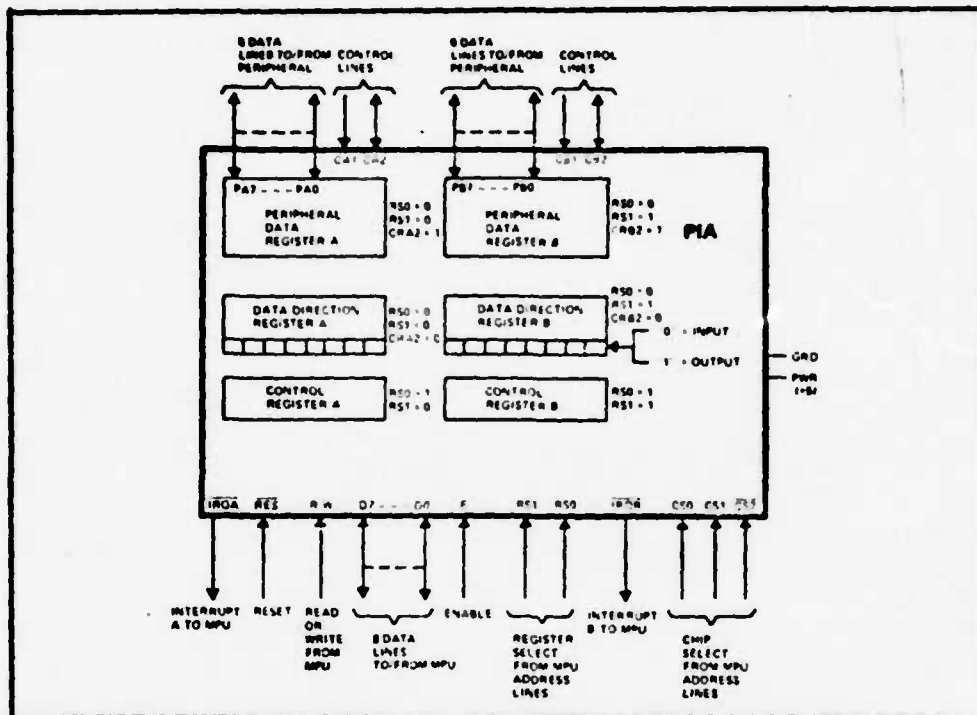


Figure 6. MC6821 Peripheral Interface Adapter (PIA) [Ref. 8]

RS1	RS0	Register Selected
0	0	Data Direction Register A (DDRA) (bit 2 of CRA = 0) Peripheral Data Register A (PDRA) (bit 2 of CRA = 1)
0	1	Control Register A (CRA)
1	0	Data Direction Register B (DDRB) (bit 2 of CRB = 0) Peripheral Data Register B (PDRB) (bit 2 of CRB = 1)
1	1	Control Register B (CRB)

Figure 7. MC6821 Peripheral Interface Adapter (PIA) Register Selection [Ref. 8]

the signals from all three of the thermometers present in the sample holder. Before this interface could be designed, however, one important design decision had to be made.

As was stated in the last chapter, two different schemes could be used for biasing the control thermometers (the carbon-glass and platinum resistors) and measuring the resulting signals: applying a constant voltage and measuring the resulting current, or applying a constant current and measuring the resulting voltage. A decision was made at this point as to which of these two schemes to use. In examining the data sheet on the carbon-glass thermometer, it was found that the maximum recommended operating voltage was only 10 millivolts. In addition, the resistance of this thermometer changes by two orders of magnitude over its usable range (Fig. 3). This presented problems with the constant current biasing scheme. Namely, designing a constant current source that could remain constant with such a large variation in load resistance. Similar problems were also evident with the platinum resistor. The use of a constant voltage source scheme, however, presented very few problems. Therefore, it was decided that the control thermometers were to be biased with a 10 millivolt constant voltage source (Fig. 8) and the resulting current was to be measured.

The next obvious step was to devise a method of measuring the current through each of the control thermometers. This problem was solved by examining a classical inverting amplifier circuit (Fig. 9). The equation for the output voltage is given by:

$$V_o = - R_f/R_i * V_i \quad (3)$$

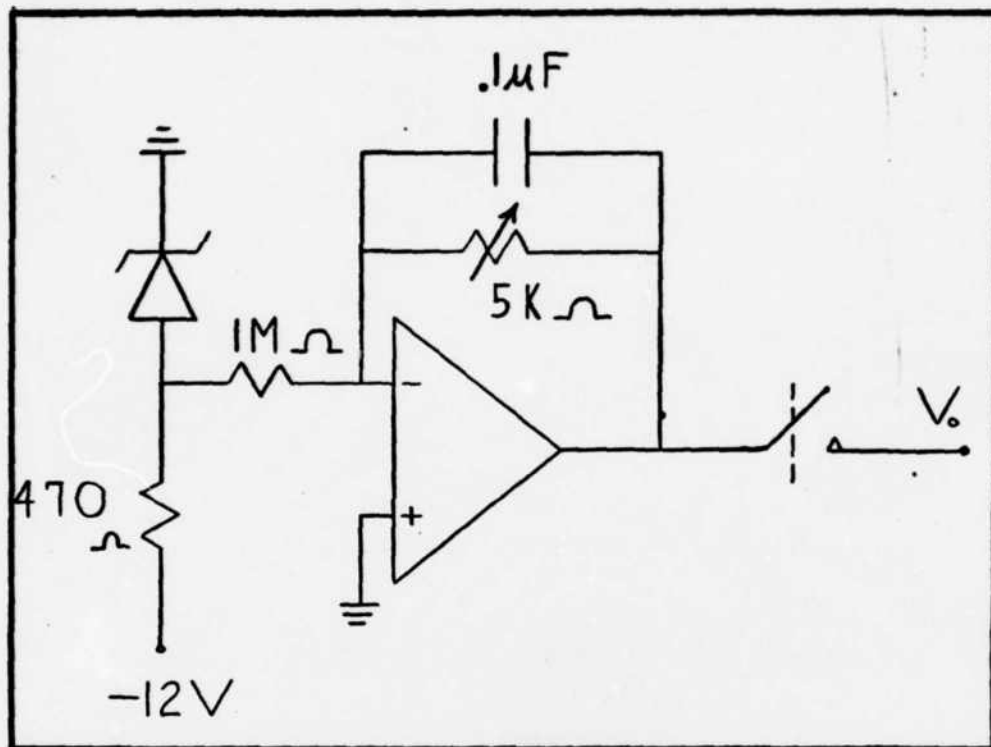


Figure 8. Control Thermometer Constant Voltage Biasing Source

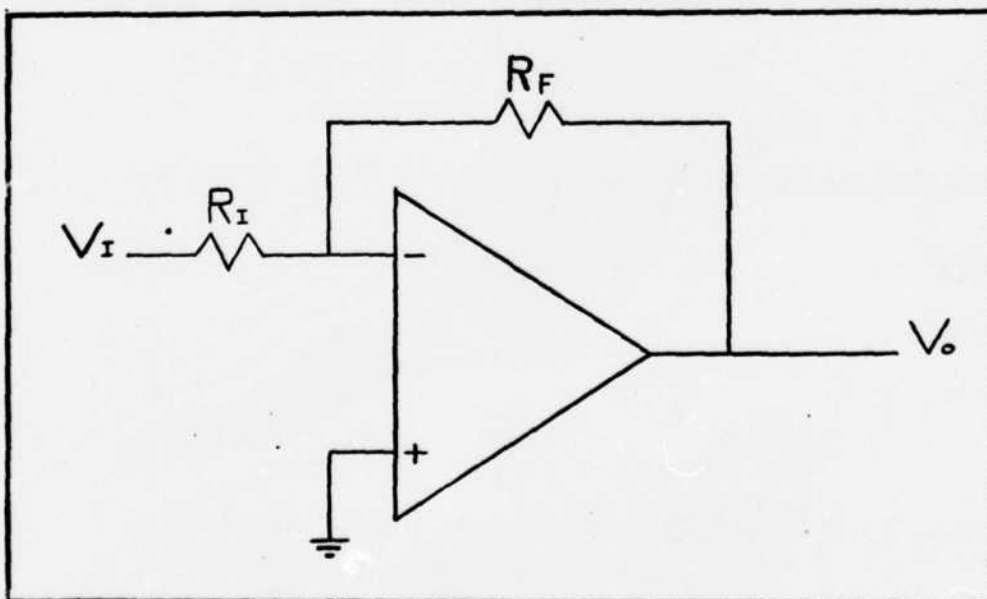


Figure 9. Classical Inverting Amplifier

By considering the input voltage to be the constant voltage source and the input resistance to be the resistance of the control thermometer, it was evident that the output voltage was equal to the current through the control thermometer multiplied by the feedback resistance. The voltage could then be converted to a digital signal by using a standard analog-to-digital (A/D) converter. By modifying the inverting amplifier circuit as shown in Figure 10, the amplifier gain can be programmed and the input switched from one control thermometer to another.

The only remaining thermometer to be interfaced to the microprocessor system was the silicon diode. As stated earlier, this thermometer produced a voltage signal and therefore could be connected directly to an A/D converter. However, the user was concerned about possible loading effects of the automatic temperature controller on the measurement of the silicon diode voltage by the AHEEDAS system. Therefore, it was decided that the input from the silicon diode would be buffered by a voltage follower amplifier stage. In addition, to insure that the automatic temperature controller did not affect voltage measurement by the AHEEDAS, a Coto-coil Model U-20146 Relay was also placed on the input to the automatic temperature controller. This relay had an off-state impedance of 100 gigohms [Ref. 9] and would only be closed during times when the automatic temperature controller needed to monitor the silicon diode voltage (while changing temperature). In addition to isolating the silicon diode from the automatic temperature controller, it was also desirable to have the same programmable gain capability to measure the diode voltage as was already established for the control thermometers. This was accomplished by feeding the output of the voltage follower through an analog switch and an input resistor and into the

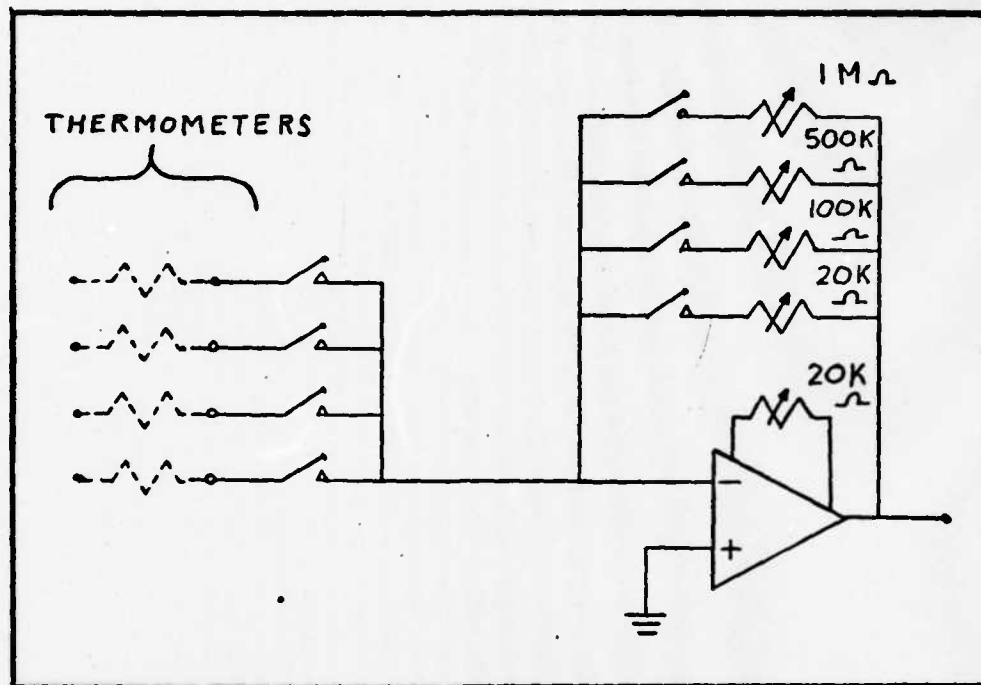


Figure 10. Modified Inverting Amplifier

programmable gain amplifier of the current sensing stage. Therefore, the output signal from each of the thermometers could be monitored at the output of the programmable gain amplifier by closing the appropriate switches.

After determining how the signals from each of the thermometers could be detected, it was time to generalize the detection scheme so that the automatic temperature controller could be used on future projects. Since it was possible for a future project to require more than three thermometers and since it was also possible that these thermometers could produce either voltage or current signals, the design of the automatic temperature controller was modified to account for these possibilities. An arbitrary choice of a maximum of four thermometers

(a power of two) was made. By adding relays to both the voltage signal inputs and the current signal inputs, the original design could handle the additional thermometer.

One further modification was made to the original design. It was decided that there may be times when the input to the A/D converter should be zeroed. Therefore, one additional stage was added to the design. This stage contained a unity gain inverting amplifier and a switch. With the switch closed, the amplifier simply inverted the inverted signal from the programmable gain stage. With the switch open, the output of the last stage was zeroed. The final design of the detection section of the thermometer interface is shown in Figure 11.

The components used to build the detection section of the thermometer interface were very important to the success of the automatic temperature controller. Since any errors in detection of the thermometer signals would undoubtedly propagate, the parts chosen had to minimize these errors. Because of the small signals that had to be detected, many of the problems associated with operational amplifier design became extremely important: input bias current, input impedance, and gain. The device that seemed to have the best specifications was the LF355A JFET-input operational amplifier. This device has an input impedance of 100 megaohms and an input bias current of less than 50 picoamperes [Ref. 10]. In addition to the operational amplifier chosen, all potentiometers in the detection circuit were specified to be 15-turn potentiometers in order that each could be precisely set.

Once all the design decisions were made about the detection circuitry, the remaining component of the thermometer interface, the A/D converter, had to be specified. As stated in Chapter II, the worst

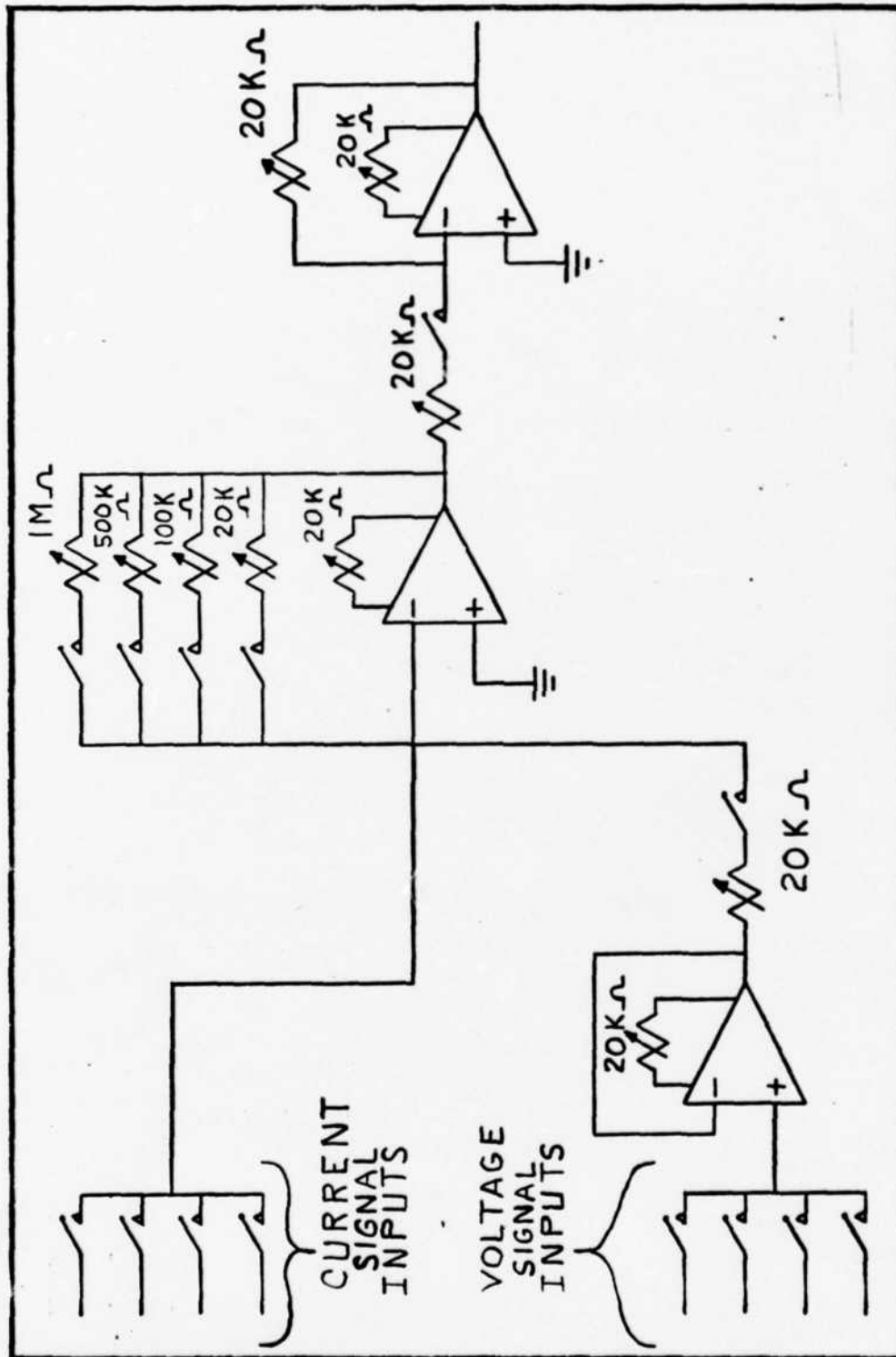


Figure 11. Thermometer Interface Detection Section

case for thermometer input signal detection was one part in 30,000. Therefore, a 16-bit A/D converter was needed in order to give the required precision. Very few 16-bit A/D converters are available, however. One A/D converter, the Intersil ICL7104-16, appeared to be appropriate for use in the thermometer interface. The ICL7104-16, one of the fastest 16-bit converters available, could do a conversion in 587 milliseconds [Ref. 11]. This seemed like it would be fast enough to give an accurate sample of the thermometer signal, yet slow enough to allow the microprocessor to do its processing before the next sample became available. The ICL7104-16 did require some external circuitry, however. In particular, an external voltage reference was required. The reference voltage circuit is shown in Figure 12.

To determine what the A/D reference voltage should be, the input signal strengths from each of the thermometers were examined. It was determined that the platinum thermometer would produce the maximum signal that the A/D converter would have to handle. The current passing through this thermometer and into the detection section of the thermometer interface produced a maximum voltage of 7.14 volts at the input of the A/D. Therefore, it was determined that a full scale reading of 8 volts on the A/D converter would be adequate for the automatic temperature controller. This corresponded to a reference voltage of 4 volts.

After the A/D converter was specified, the only remaining task to interface the thermometers to the microprocessor system was to make the appropriate connections to the PIA. In this case two PIAs were required: one to handle the 16 data lines of the A/D converter, and another to handle the lines to the analog switches, relays, and A/D

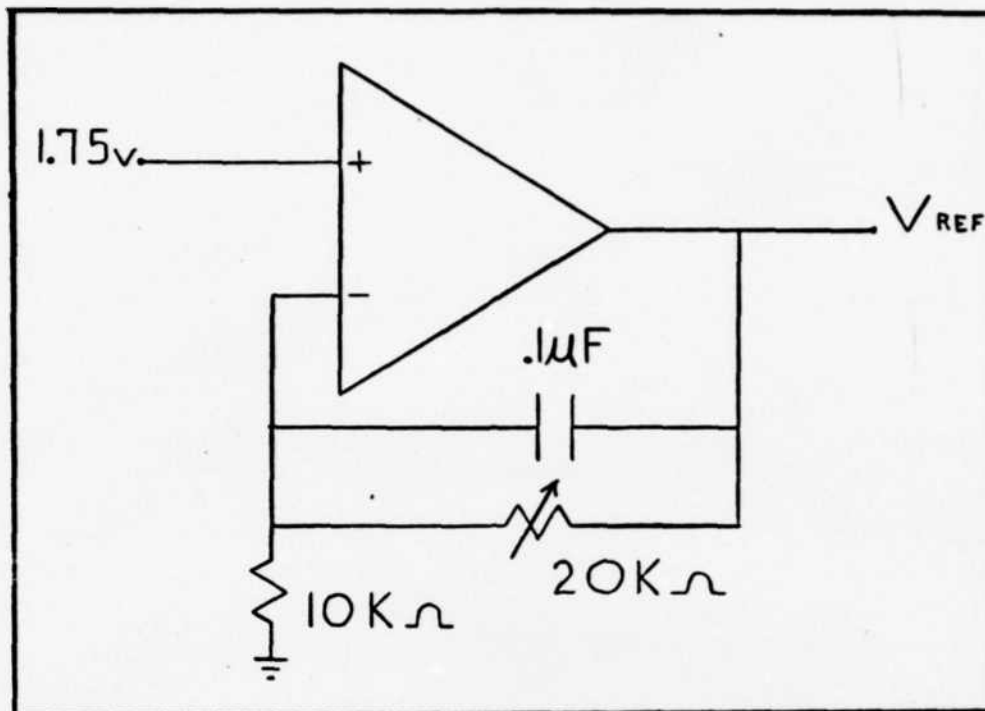


Figure 12. A/D Reference Voltage Circuit

control and status pins. These connections are explained in more detail in Appendix C.

Heater Coil Interface

After the thermometer interface design was completed, the remaining interface to the sample holder, the heater coil interface, had to be designed. The heater coil was required to be driven by the HP 6130C Digital Voltage Source. Therefore, this voltage source had to be interfaced to the microprocessor system. Before this interface could be completed, however, an examination of the HP 6130C was needed.

The HP 6130C Digital Voltage Source is a complete digital-to-analog link between a computer and any application requiring a fast,

accurate source of dc power. The output voltage is controlled by 17 digital data inputs applied to the HP 6130C via a ribbon connector on the rear panel. In addition, overcurrent protection is provided by a current latch circuit which can be externally programmed to one of eight values between 2 percent and 100 percent of the units rated output of 1 ampere (Fig. 13). The current limit and voltage data is latched into the HP 6130C by a gating signal provided by the computer. The HP 6130C also provides signals back to the computer. The status outputs of the HP 6130C inform the computer of overload conditions, current limit latch status, and busy states. [Ref. 12]

In order to complete the heater coil interface, each of the lines mentioned in the previous paragraph had to be connected to the microprocessor system. This required the use of 2 PIAs: one to handle 16 of the voltage data lines, and one side of another PIA to handle the remaining voltage data line, the current limit data lines, and the status lines from the HP 6130C (Fig. 14). The gating signal was provided by using an SN74LS123 monostable in conjunction with one of the control lines from the PIAs. These connections completed the heater coil interface design and, therefore, all the connections between the sample holder and the automatic temperature controller.

LSI-11 Computer Interface

The LSI-11 computer interface was the only remaining interface to the experimental apparatus. But, before this interface could be designed, the information that would be exchanged between the LSI-11 computer and the automatic temperature controller had to be determined. Since it was known that the LSI-11 contained the voltage/temperature

Output Line Logic Levels			Current Limit (milliamps)
L24	L23	L22	
1	1	1	20
1	1	0	50
1	0	1	70
1	0	0	100
0	1	1	200
0	1	0	500
0	0	1	700
0	0	0	1000

Figure 13. HP 6130C Current Limit Data Format

conversion table for the silicon diode thermometer, the LSI-11 could transmit either the desired temperature value or the corresponding voltage to the automatic temperature controller. Sending the voltage value, however, would eliminate the need for the temperature controller to also have a voltage/temperature conversion table. Therefore, data only needed to be transmitted from the LSI-11 to the automatic temperature controller. The only information the LSI-11 required of the automatic temperature controller was when the temperature had been set to the desired value. This could be accomplished by using a single line.

After the signal interface was defined, the next task in the design of the computer interface was to identify how the LSI-11 communicated to the outside world. This was done through a DRV-11 parallel interface card [Ref. 13]. The DRV-11 is very similar to the MC6821 PIA and therefore the connections between these two devices were fairly straight-

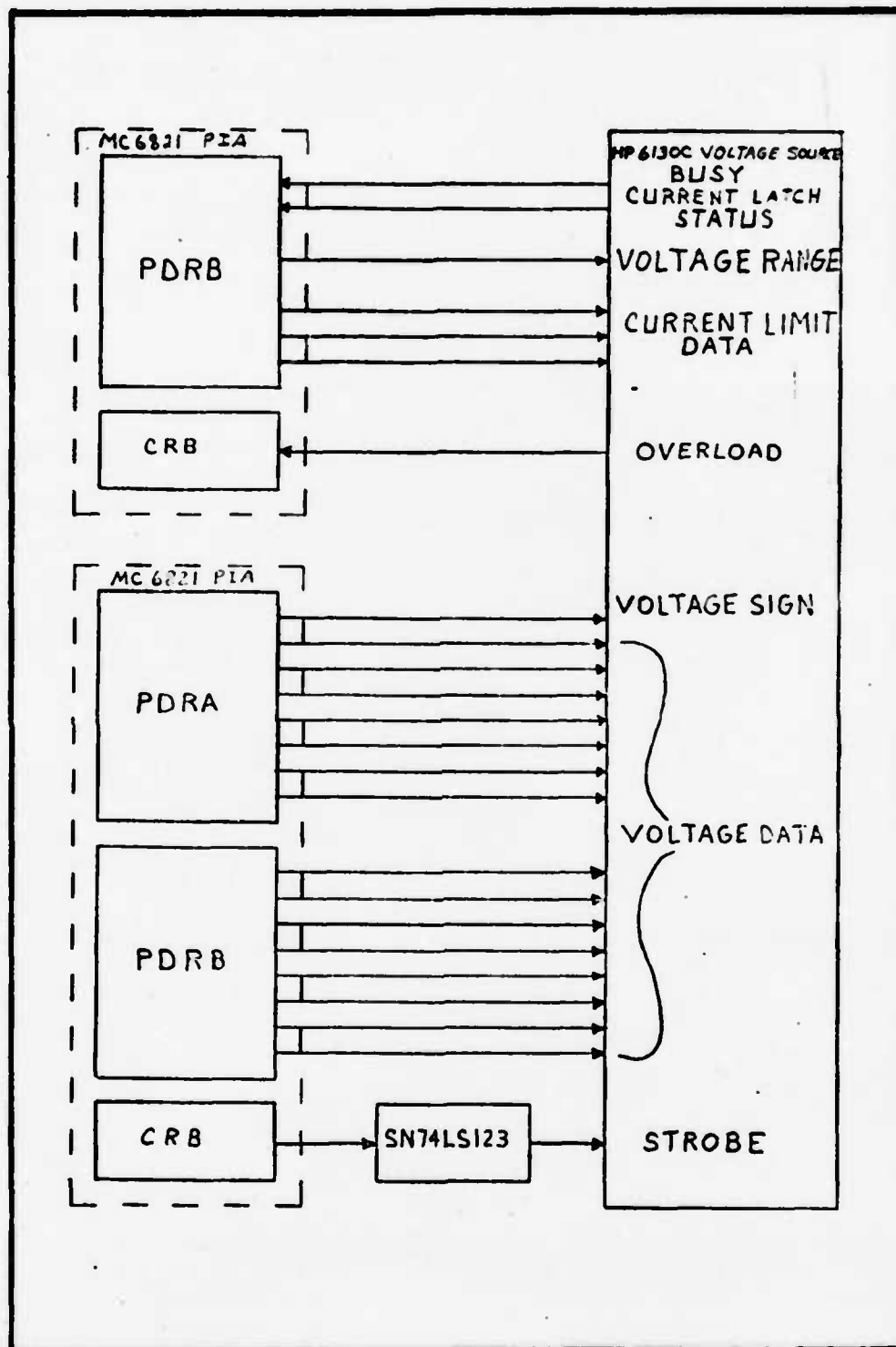


Figure 14. Heater Coil Interface

forward. The LSI-11 could transmit data 8-bits or 16-bits at a time. It was decided to send the data 16-bits at a time, thereby increasing the speed of the data transfer. Two control lines on the PIA directly interfaced with two of the control lines on the DRV-11. Therefore, the handshaking signals needed to control the data exchange could easily be established. The remaining signal, the temperature set signal, was to be generated by the automatic temperature controller by using one of the two remaining control lines of the PIA. This line could also be connected directly to the DRV-11.

These connections between the MC6821 PIA and the DRV-11 completed the design of the interface between the microprocessor system and the LSI-11 computer.

Display Interface

Once the interfaces to the existing experimental equipment and the LSI-11 computer were designed, the interfaces to the input and output devices of the automatic temperature controller had to be designed. The first of these devices was the display. As was stated in the requirements, the display needed to be an alphanumeric display so that the user could be prompted for input and monitor the execution of the controller program. It was decided that a 16-digit display would be adequate for this purpose. To meet this requirement, the Hewlett-Packard HDSP-6508 16-segment Alphanumeric LED Display was chosen. The HDSP-6508 is a low power device with common cathode digit connections and common anode segment connections. In addition, the HDSP-6508 is an eight digit device [Ref. 14]. Therefore, two of these devices were needed to form a sixteen digit display. It was clear that both display

drivers and data multiplexors would be needed in order to integrate the HDSP-6508 into the design of the temperature controller. Before work was begun on designing this circuitry, an overall look at the display interface was required.

With the design philosophy in mind, it was decided that the character decoding should be done in hardware rather than in software. This would relieve quite a bit of the work burden placed on the microprocessor. Since the simplest way to implement the character decoding in hardware was to use a ROM look-up table, a character set could be devised that would further reduce the work burden on the microprocessor. By assigning the hex codes 00 through 0F to the hex digits 0 through F, hexadecimal numeric data could be sent directly, digit-by-digit, to the display port. The complete character set is shown in Figure 15.

Once it was determined how the characters would be decoded, the next step was to determine how the microprocessor would transmit characters to the display. One way was to use one PIA. Since the MC6821 PIA has two separate ports, one port could be used for outputting a character and the other could be used for identifying where in the display the character should be placed. Other methods were examined, but none were as esthetically pleasing as this method. Therefore, it was decided to use only one PIA to interface the microprocessor system with the display.

Up to this point, it was determined that the microprocessor system would interface with the HDSP-6508 display through one PIA and that the characters would be decoded in hardware. The next step was to design the additional display interface hardware that would make it all work. Since only one digit could appear at the output ports of the PIA at a

Character	Hex Code	Character	Hex Code	Character	Hex Code
0	00	O	18		2F
1	01	P	19	space	30
2	02	Q	1A	↑	31
3	03	R	1B	"	32
4	04	S	1C	↓	33
5	05	T	1D	\$	34
6	06	U	1E	%	35
7	07	V	1F	&	36
8	08	W	20	'	37
9	09	X	21	(38
A	0A	Y	22)	39
B	0B	Z	23	*	3A
C	0C	[24	+	3B
D	0D]	25	,	3C
E	0E	<	26	=	3D
F	0F	>	27	Σ	3E
G	10	=	28	Δ	3F
H	11	?	29	÷	68
I	12	[2A	;	7C
J	13	/	2B	.	80
K	14]	2C	!	B7
L	15	^	2D	:	FO
M	16		2E	+	FD
N	17				

Figure 15. Automatic Temperature Controller Character Set

time, it was necessary to have some sort of storage capability in the display interface. If this was not done, the microprocessor would be required to constantly update the display. Therefore, an MCM6810 RAM was included in this interface and circuitry was developed so that when the microprocessor output a character to the PIA, the character would be stored in the RAM. After this circuitry was developed, additional circuitry was designed that continuously read each of the characters from RAM, decoded these characters, and drove the proper digits and segments corresponding to each of these characters. A complete diagram of the display interface is shown in Figure 16.

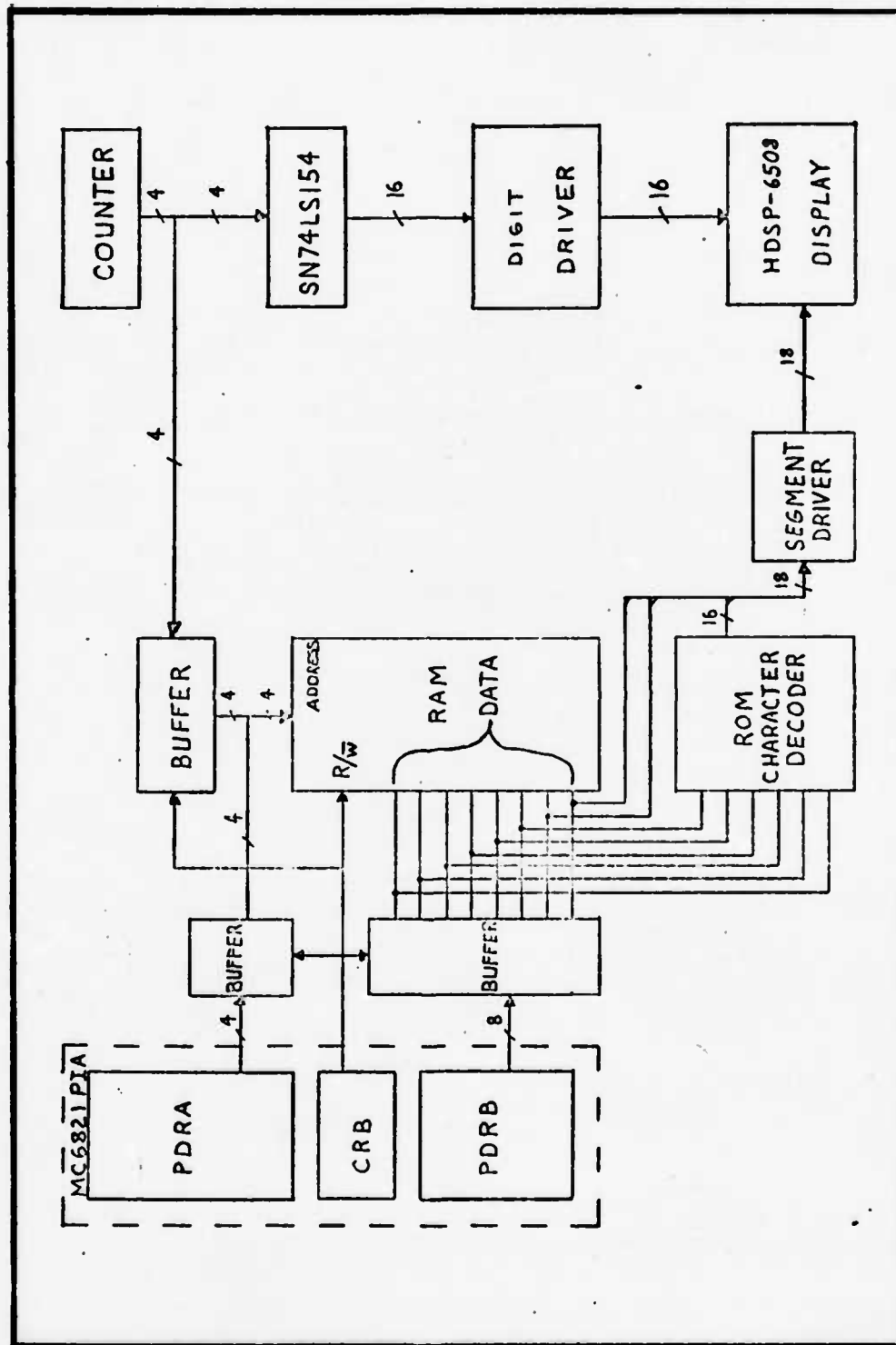


Figure 16. Display Interface

Switch and Keypad Interface

Once the interface to the display had been designed, the remaining items, the input devices, had to be interfaced to the microprocessor system. The keypad, as stated in Chapter II, was required so that the user could enter data into the controller. In addition, two other items were included in the design of the automatic temperature controller: a toggle switch, and a rotary switch. The toggle switch was used to indicate to the microprocessor system whether the controller was to be used in an automatic mode under the control of the LSI-11 or in a manual mode under control of the user. The rotary switch was used to select the thermometer to be used for control. Since four thermometers were allowed for in the design of the thermometer interface, four positions of the rotary switch would correspond to each of these thermometers. In addition, one other position was used to indicate to the microprocessor system that the control thermometer was to be selected by the automatic temperature controller. The keypad, the toggle switch, and the rotary switch, therefore, all had to be interfaced to the microprocessor system to make the hardware design complete.

Since the keypad, the toggle switch, and the rotary switch are all mechanical devices, the signals from these devices contained noise due to bouncing of the mechanical contacts. Therefore, these signals had to be debounced. Two possible methods of signal debouncing were available: software debouncing and hardware debouncing. Again, the hardware approach was chosen so as not to burden the software with processes that could easily be done in hardware. Therefore, the next task was to choose a signal debouncer. The Motorola MC14490 Hex Contact Bounce Eliminator was chosen to perform this function. This

device is capable of debouncing six signal lines. Also, each input to this device is equipped with an internal pull-up resistor thereby eliminating the need for external pull-up resistors. [Ref. 15] A total of fourteen lines -- eight from the keypad, five from the rotary switch, and one from the toggle switch -- needed to be debounced. Therefore, three MC14490 packages were required.

After determining how the signals from the input devices could be debounced, the next task was to interface these lines to the microprocessor. Again, the MC6821 PIA was used to perform this function. One data port of the PIA was devoted to the eight lines from the keypad. The remaining port was used to interface the lines from the toggle and rotary switches. The only remaining circuitry that had to be developed was the circuitry to indicate to the microprocessor that data is available at the input ports of the PIA. This was accomplished by using exclusive-or gates connected to the data lines from the keypad and switches. Any change in switch status or any depression of a key would generate a change in state of the exclusive-or gate output. By connecting this signal to one of the control lines of the PIA, the microprocessor could be alerted to new data at the PIA.

A complete diagram of the switch and keypad interface is shown in Figure 17.

Summary of Hardware Design

At this point in the hardware design, the interfaces had been designed for each of the peripheral devices: the thermometers, the heater coil, the LSI-11 computer, the display, the keypad, and the switches. All of these interfaces required a total of 7 MC6821 PIAs.

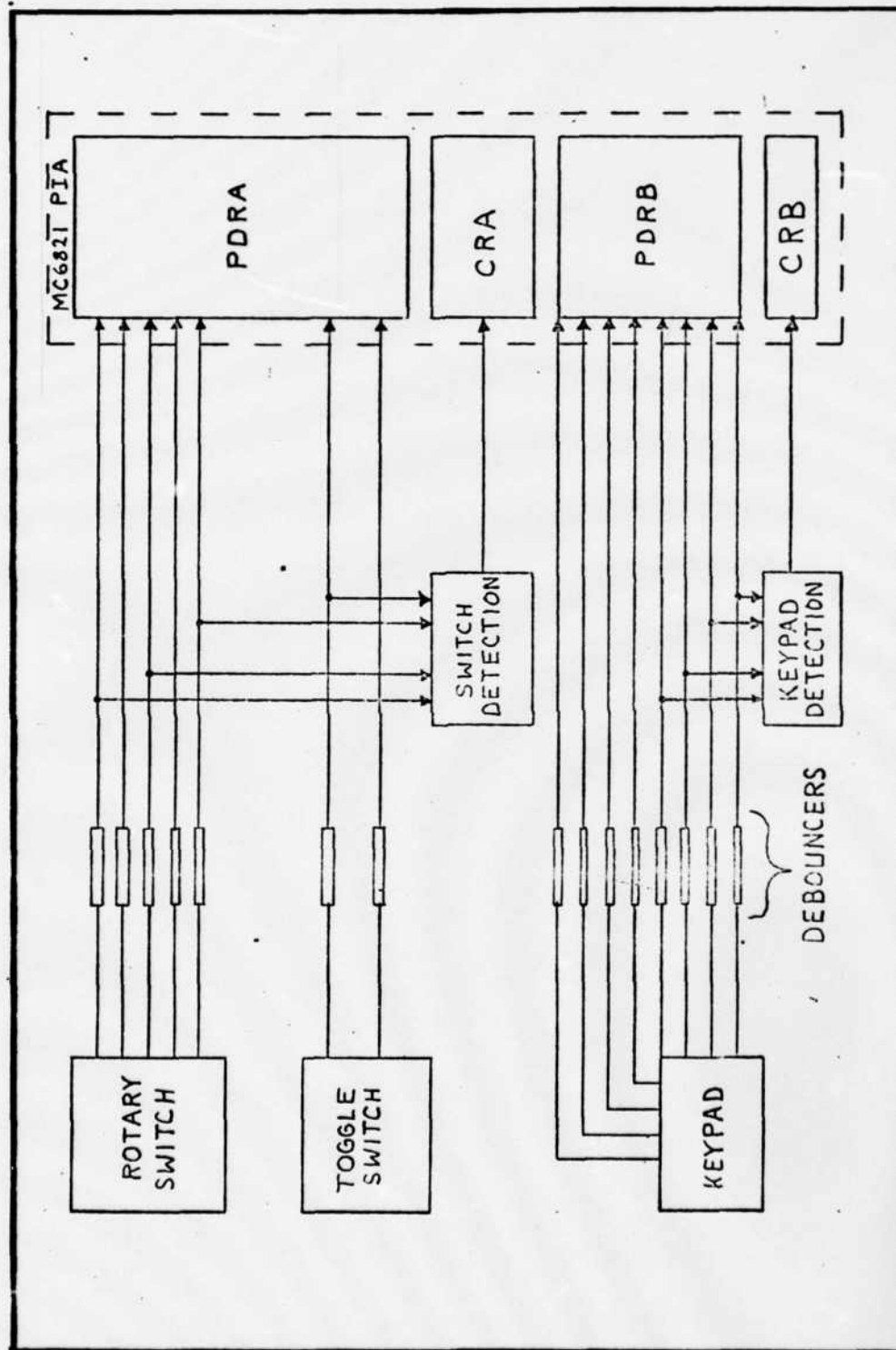


Figure 17. Switch and Keypad Interface

Added to the 7 PIAs were the 10 memory chips of the microprocessor system yielding a total of 17 devices that were to be placed on the address and data busses of the MC6809 microprocessor. To enable the microprocessor to drive each of these devices, both the address and data busses were buffered with standard TTL buffer chips. This not only enabled the MC6809 microprocessor to drive the devices designed to be on the busses, but also allowed for possible expansion of the microprocessor system.

The final task of the hardware design was to decide where the memory, input/output ports, and input/output control registers would reside in the address space (Fig. 18). The 640 bytes of RAM were chosen to reside in locations 0000 to 027F and the 10K bytes of ROM were chosen to reside in locations D800 to FFFF. This left only the PIA data and control registers to be placed in the address space of the microprocessor. It was decided to place all the PIA data register in contiguous memory locations beginning with 1000 and to place all the PIA control registers in contiguous memory locations beginning with 1010 so that data and control register accesses could be more easily identified during the hardware testing. To do this, address lines A4 and A0 were connected to RS0 and RS1 of the PIAs and address lines A1, A2, and A3 were decoded to provide the chip selects for each of the 7 PIAs.

Address decoding was done using SN74LS138 Decoders. While a fully-decoded addressing scheme was not used, ample additional decoder outputs were provided so that an additional 2K bytes of ROM, 384 bytes of RAM, and one PIA could be added to the existing hardware with no additional decoding hardware being needed.

Memory Locations	Device
0000 - 007F	RAM1
0080 - 00FF	RAM2
0100 - 017F	RAM3
0180 - 01FF	RAM4
0200 - 027F	RAM5
1000 - 1001	PIA1-DATA
1002 - 1003	PIA2-DATA
1004 - 1005	PIA3-DATA
1006 - 1007	PIA4-DATA
1008 - 1009	PIA5-DATA
100A - 100B	PIA6-DATA
100C - 100D	PIA7-DATA
1010 - 1011	PIA1-CONTROL
1012 - 1013	PIA2-CONTROL
1014 - 1015	PIA3-CONTROL
1016 - 1017	PIA4-CONTROL
1018 - 101A	PIA5-CONTROL
101A - 101B	PIA6-CONTROL
101C - 101D	PIA7-CONTROL
D800 - DFFF	EPROM1
E000 - E7FF	EPROM2
E800 - EFFF	EPROM3
F000 - F7FF	EPROM4
F800 - FFFF	EPROM5

Figure 18. Microprocessor System Memory Map

At the completion of the hardware design of the automatic temperature controller, all the hardware components needed to realize an automatic temperature controller were designed. Yet, enough flexibility was left in the design to allow for modifications and additions to the hardware. Complete board layouts and pinout connections are given in Appendix A. Once the hardware design was completed, the next task was the development of the software. This is the subject of the next chapter.

IV. Software Design and Implementation

After the automatic temperature controller hardware was designed, it was possible to begin writing the software needed to complete the controller. This chapter describes the use of the MC6809 microprocessor architecture, the supporting routines used to interact with the peripheral equipment, the operating scenario of the temperature controller, and the development of the main program that actually performed the temperature control.

Use of the MC6809 Architecture

The first decision of the software design was to determine how the architecture of the MC6809 microprocessor was to be used. The MC6809, as stated in the previous chapter, has two stack pointers, two index pointers, and three interrupt lines. The two stack pointers, the hardware stack pointer and the user stack pointer, are both 16-bit registers which contain the memory addresses of the top of each of the respective data stacks. The hardware stack is used by the microprocessor to save the return address during a subroutine call and to save the processor status during an interrupt request. The hardware stack can be used by the user. However, care must be taken to insure that all data that are pushed onto this stack are pulled from this stack before a return from subroutine or a return from interrupt is issued. The user stack, on the other hand, is completely at the disposal of the user. [Ref. 7] Since it was anticipated that a great deal of number crunching would

have to be done by the automatic temperature controller, it was decided that the user stack should be used for performing all arithmetic calculations. In addition, since this stack would not be affected by subroutine calls and interrupts, the user stack was to be used for passing parameters to the subroutines. The hardware stack, on the other hand, was to be used for storing loop counters and in any situation where it was inconvenient to use the user stack.

The two index pointer registers are each 16-bit registers that may be used to point to any memory location. It was decided that these two registers could be used as pointers to generalize some of the software routines. For example, a routine to compare two numbers could assume that these two pointers pointed to the numbers to be compared. Therefore, the numbers themselves would not have to be passed to the routine. In addition to this use, it was decided that these registers could also be used as temporary storage and as counters for long loops.

The final component of the MC6809 architecture whose use was to be determined was the interrupt lines. Three interrupt lines are available: the normal interrupt line, the fast interrupt line, and the non-maskable interrupt line. It was determined that the normal interrupt line was to be used to handle interrupts from each of the following sources: the HP 6130C Digital Voltage Source, the LSI-11 computer, the keypad, and the switches. This left the use of the fast interrupt line and the non-maskable interrupt line to be determined. It was not anticipated that either of these lines would be needed for the normal operation of the temperature controller. Therefore, these two interrupt lines were used to aid in the hardware testing of the automatic temperature controller. The non-maskable interrupt line was used to trigger execution

of a program that would enable the user to examine all of the registers of the microprocessor as they were at the time of the interrupt. This capability along with a data analyzer served as a development system for the automatic temperature controller. The fast interrupt line, on the other hand, was used to trigger execution of a program that checked the integrity of the arithmetic subroutines.

Supporting Software

Once it was determined how the various structures of the MC6809 microprocessor architecture were to be used, the next task was to begin the design of the software. Since it was known that the controller algorithm would have to interact with each of the peripheral devices and that it undoubtedly would require the use of basic arithmetic routines, the first chore of the software development was to write subroutines that would perform these functions. These subroutines were broken into seven categories: display routines, switch interaction routines, computer interaction routines, keypad interaction routines, voltage source interaction routines, thermometer interface interaction routines, and arithmetic routines. A short description of each of these routine types is given below. A more complete description of these routines is given in Appendix B.

Two types of display routines were developed. The first type is based upon a subroutine that simply outputs a character to the display. It was anticipated, however, that this type of routine would not be sufficient for the controller algorithm, especially if long prompts were required to be displayed. Therefore, a second set of display subroutines was also developed. These subroutines are based upon a subroutine that scrolls characters onto the display. Each of these two sub-

routine types contains routines which display a character, a string of characters, a byte of information in hexadecimal, a floating point number in hexadecimal, and a floating point in decimal. In addition, a subroutine to blank the display is also included.

The second category of subroutines, switch interaction routines, should more correctly be called switch interaction routine. It was decided that only one routine was needed to perform all the interaction between the microprocessor system and the rotary and toggle switches. This routine reads the current status of the rotary and toggle switches and updates the memory locations where this status is stored and also modifies the interrupt capability of the keypad and the LSI-11 computer to correspond with the present position of the toggle switch (AUTO/MAN).

The third category of subroutines is a set of routines used to interact with the LSI-11 computer. Two functions had to be implemented here. The first was to read information being sent from the LSI-11 computer and convert this information to a form that could be used by the automatic temperature controller. The second function was to send a signal back to the LSI-11 indicating whether the temperature was set or not. This function, however, required only a load and store and it was felt that a subroutine was not warranted for only two instructions. Therefore, no subroutine is written to perform the second function.

The fourth category of subroutines are the keypad interface routines. It was anticipated that three forms of input would be required of the keypad: (1) a single decimal digit, (2) a yes or no answer, and (3) a real number. In addition, it was also anticipated that the user could possibly make mistakes when entering information or want an extremely long prompt repeated. Therefore, each of these routines had

to display the input as it was entered and allow for correcting the input or repeating the prompt. Based on this, the keypad was 'painted' as shown in Figure 19 and a routine to interpret the row and column signals from the keypad was written. Using this routine then, the three input routines could be constructed.

Three subroutines are written for the fifth category of routines, the voltage source interfacing routines. These include routines which set the output current limit, set the output voltage, and read the output voltage setting of the HP 6130C Digital Voltage Source.

The thermometer interfacing routines make up the sixth category of subroutines. This category consists of two subroutines: one which places the bias voltage on all the sensors which require a biasing voltage from the automatic temperature controller, and another which reads the data supplied by the A/D converter and adjusts the gain of the programmable gain amplifier. It was decided that thermometer selection would best be done by the main program and, therefore, a subroutine to perform this function is not provided.

The arithmetic routines comprise the final category of subroutines. These routines are the ones that handle the arithmetic manipulation of floating point numbers. Routines to recall, store, add, multiply, negate, and invert floating point numbers have been written. To these routines have been added routines that perform stack manipulations and compare real numbers. These routines, it was felt, would be sufficient to implement the automatic temperature controller.

Once all the subroutines were written, it was possible to also write the interrupt handler routines. These routines are also contained in Appendix B.

CLEAR	7	8	9
YES	4	5	6
NO	1	2	3
REPEAT	0	•	↑ ENTER

Figure 19. Keypad Layout

Control Scenario

After the supporting software was written, the next step was to determine how the automatic temperature controller was to adjust and control the temperature of the sample holder. To meet this end, a scenario was developed that closely mimicked the actions of the technician and the Artronix analog controller in the original temperature control scheme.

The scenario for the automatic temperature controller was envisioned as follows: The automatic temperature controller would read the

desired silicon diode voltage setting from the LSI-11 computer or the keypad. Which of these two devices would be read would depend upon whether the automatic temperature controller was in the automatic mode or in the manual mode. This would be determined by the position of the toggle switch. Once the desired voltage was obtained, the controller would determine which control thermometer should be used when the desired voltage was reached. The controller would base its decision upon the position of the rotary switch, the desired voltage, and information previously supplied by the user on each of the control thermometers. After the appropriate control thermometer had been selected, the controller would begin to monitor the silicon diode thermometer voltage and adjust the output of the voltage source to obtain the desired voltage on the silicon diode. This adjustment would be controlled by the temperature controller so that the desired voltage would be obtained as quickly as possible without overshooting the desired voltage set point. Once the controller determined that the voltage was set and that it was stable, the controller would switch to the appropriate control thermometer and signal the LSI-11 computer that the temperature was set. The automatic temperature controller would then continue to monitor the control thermometer, making adjustments as necessary to the voltage output to the heater coil in order to maintain the control thermometer reading. This temperature control function would continue until the controller was interrupted with the next input from the computer or the keypad.

Control Scheme

Once the control scenario was devised, the next step in the software development was to implement this scenario. The most difficult

portion of the scenario to implement is the changing of temperature. Therefore, this was the first portion to be implemented.

As mentioned in the discussion on temperature adjustment in Chapter II, the adjustment of temperature became difficult below approximately 40 degrees Kelvin due to the change in thermalconductivity of the copper sample holder. Since the copper sample holder was not changed in the hardware design of the controller, temperature adjustment for the automatic temperature controller would also be difficult. Therefore, the method used to adjust temperature would have to account for this change in thermalconductivity. Since the automatic temperature controller would be reading the input from the silicon diode and adjusting the heater coil voltage source accordingly, all information necessary to derive the impulse response of the thermal system, as seen by the controller, would be known. Furthermore, with the impulse response in hand, the controller could accurately predict what heater coil voltage would produce a desired input voltage from the silicon diode. Therefore, the problem of temperature adjustment can be solved by deriving and implementing an equation that gives the heater coil voltage in terms of a desired input thermometer signal level and the impulse response of the thermal system.

An equation that related the desired input signal level and the output voltage was derived and implemented. However, late in the testing of the automatic temperature controller, this equation was found to be in error. This error is still reflected in the source listing contained in Appendix B and will be discussed in Chapter V. A similar, but more correct derivation of this equation is given in the paragraphs below.

Let:

$y(n)$ = nth sample of the input signal from the thermometer

$x(n)$ = nth sample of the output voltage to the heater coil

$h(n)$ = nth sample of the impulse response of the thermal system
as seen by the automatic temperature controller

These quantities are related by the convolution sum:

$$y(n) = \sum_{k=0}^{\infty} h(k) * x(n-k) \quad (4)$$

Assuming that $h(n)$ is equal to zero for $n \geq m > 0$, equation (4) can be expressed as:

$$y(n) = \sum_{k=0}^{m-1} h(k) * x(n-k) \quad (5)$$

Therefore:

$$\begin{bmatrix} y(n-m+1) \\ y(n-m+2) \\ \vdots \\ y(n-1) \\ y(n) \end{bmatrix}_{m \times 1} = \begin{bmatrix} x(n-m+1) & x(n-m) & \dots & x(n-2m+1) \\ x(n-m+2) & x(n-m+1) & \dots & x(n-2m+2) \\ \vdots & \vdots & \ddots & \vdots \\ x(n-1) & x(n-2) & \dots & x(n-m) \\ x(n) & x(n-1) & \dots & x(n-m+1) \end{bmatrix}_{m \times m} * \begin{bmatrix} h(0) \\ h(1) \\ \vdots \\ h(m-2) \\ h(m-1) \end{bmatrix}_{m \times 1} \quad (6)$$

Rewriting equation (6) using matrix variables gives:

$$Y = X * H \quad (7)$$

Solving for H , the impulse response, yields:

$$H = X^{-1} * Y \quad (8)$$

Since all quantities in X and Y are known to the automatic temperature controller, the impulse response, H , can be measured.

The next step in the derivation was to determine the proper heater coil voltage setting to give a desired input signal from the thermometer. To do this, it was assumed that at some time, p , the desired input signal level, $y(p)$, is known. Therefore, the heater coil voltage, $x(p)$, can be determined by substitution into equation (5). Solving for $x(p)$ yields:

$$x(p) = \frac{y(p) - ([0 \ x(p-1) \ x(p-2) \ \dots \ x(p-m+1)] * H)}{[1 \ 0 \ 0 \ \dots \ 0] * H} \quad (9)$$

The remaining step in the derivation was to substitute for the impulse response. The most recent measurement of the impulse response was obtained by letting $n = p-1$ in equation (8). Substituting this expression into equation (9) and simplifying gives the final form of the equation:

$$x(p) = \frac{y(p) * \det(B) - \sum_{k=1}^{m-1} x(p-k) * \det(A(k))}{\det(A(1))} \quad (10)$$

where:

$$B = \begin{bmatrix} x(p-m) & x(p-m-1) & \dots & x(p-2m) \\ x(p-m+1) & x(p-m) & \dots & x(p-2m+1) \\ \vdots & \vdots & \ddots & \vdots \\ x(p-2) & x(p-3) & \dots & x(p-m-1) \\ x(p-1) & x(p-2) & \dots & x(p-m) \end{bmatrix}$$

and

$$A(k) = B \text{ with the } k\text{th column replaced with the column vector } [y(p-m) \ y(p-m+1) \ \dots \ y(p-2) \ y(p-1)]^T$$

Equation (10) can be used to determine $x(p)$ if the matrices $A(l)$ and B are not singular. If the matrix $A(l)$ is singular, the equation is undefined and if the matrix B is singular, equation (8) from which equation (10) was derived is undefined. An implementation of this equation, therefore, must insure that these matrices are not singular.

One way in which equation (10) could be implemented to insure that the two matrices would not be singular is as follows: A value of m would be chosen so that the microprocessor system could be expected to evaluate the equation in a reasonable amount of time. An attempt would be made by the microprocessor to determine the heater coil output voltage from equation (10) by first evaluating the determinants of the $A(l)$ and B matrices. If either of these determinants was zero, the microprocessor would attempt to solve the equation using $m-1$ points, still checking for singularity of the $A(l)$ and B matrices. This process could be repeated -- decrementing m and attempting to solve the equation -- until the equation was solved or until m became equal to zero. (If all samples of the heater coil voltage were zero, the determinant of the B matrix would be equal to zero for all values of m .) If the equation could not be solved, the microprocessor would output some predefined voltage level to the heater coil. (This same scheme was used in the actual implementation of the controller.)

Several assumptions have not been stated that can not be neglected in the implementation of the controller. In order to use the convolution sum as it is stated in equation (4), the thermal system must be causal, time-invariant, and linear. The first requirement, causality, is no problem, since this is a real system. The second requirement is for the system to be time-invariant. The thermal system, however,

is obviously not time-invariant, since the impulse response of the system varies with temperature and the temperature is being varied with time. It was assumed, however, that the temperature of the system will not change abruptly and therefore the system will look approximately time-invariant. Therefore, time-invariance was not overwhelming restriction. The linearity requirement, however, did present a problem in the development of the control scheme. Since the thermometers are far from being linear devices, the thermal system, as seen by the controller, is also not linear. It was assumed, however, that the thermometers would appear to be linear for small changes in temperature since their input signals change monotonically with temperature. The thermometers whose input signals are monotonically decreasing with temperature presented an additional problem. In the typical model of a linear system, a zero input signal will produce a zero output signal [Ref. equation (4)]. The thermometers with monotonically decreasing input signals, however, produce their largest signals when no voltage is applied to the heater coil. Therefore, the signals from these thermometers have to be modified so that they are smallest at lower temperatures. One method of doing this was devised, but was found to be incorrect. This method and a more correct method are discussed in Chapter V. The incorrect method stated that the reciprocal of the input samples from these thermometers was to be used instead of the actual input samples. Using this technique, the thermometers whose signals were originally decreasing with temperature were now increasing with temperature and, therefore, could be used in the temperature adjustment equation. A more correct method is to subtract the thermometer signal at the lowest achievable temperature from each of the input samples.

Using the more correct method, all of the assumptions can be satisfied. Therefore, equation (10) is a valid equation for adjusting the temperature.

Once it was determined that the assumptions could be satisfied, the next step in the software development was to decide upon a model which the temperature controller could use to provide the intermediate thermometer input levels, $y(p)$, to the control equation. The choice of model was completely arbitrary. However, the user was particularly concerned about overshooting the desired temperature. Therefore, it was decided that the automatic temperature controller should follow a model of a critically damped curve of input signal versus temperature. This model was chosen so that the desired temperature would be obtained in no fewer than twenty samples.

After the model for changing temperature was determined, it was noted that the same control algorithm used for changing temperature could also be used for controlling temperature. Assuming that when the switch is made to monitoring the control thermometer that the temperature is stable at the desired level, the first reading of the input signal from the control thermometer is the signal level that should be maintained. Therefore, by changing the model so that the input signal level of the control thermometer is maintained, the same equation can be used for controlling temperature as well as changing the temperature.

Required Information

Once the control scheme was devised, the final task of the software development was to determine what information the user would have to supply to the temperature controller so that the controller could

adjust and control temperature. It was determined that before any adjustment or control could be done by the controller, some questions on the experiment would have to be asked: how many thermometers were to be used; which thermometer would be used to adjust temperature; and over what temperature range should the remaining thermometers be used to control temperature. In addition to these questions, other questions had to be asked in order to provide information on each of the thermometers: would the input signal from each thermometer be a voltage or a current signal; would these input signals be monotonically increasing or decreasing with temperature; if the thermometer produced a current signal, would this thermometer require voltage biasing from the temperature controller or would the biasing be done externally; and if the thermometer requires biasing from the controller, does the biasing voltage need to be applied throughout the entire experiment or only over the temperature range that the thermometer would be used. Once each of these questions were answered, all the information the automatic temperature controller would need about the thermometers would be known.

The next information the controller would need was information on the heater coil, namely: the maximum current that should be allowed to flow through the heater coil and the resistance of the coil. With these two pieces of information, the current limit on the HP 6130C Voltage Source can be set and the maximum output voltage calculated.

One final piece of information would also be required of the user. Since error in measurement is always present due to noise and other sources, the maximum allowed steady-state error for temperature adjustment would have to be specified by the user. This information allows the controller to decide when the temperature is close enough to switch

to a control thermometer.

After the required controller inputs from the user were determined, a routine was written to prompt the user for this information and store these data in the controller memory. Also included in this routine is a limited amount of error checking to insure that the input data looks reasonable. This routine is accessed upon power up of the microprocessor system.

Summary of Software Design

Once the software design was completed and implemented, the automatic temperature controller was complete and ready for testing. The next chapter discusses the results of this testing and the recommendations for how the controller can be improved.

V. Recommendations and Conclusion

Upon completion of the hardware and software designs, the automatic temperature controller was implemented and tested. This chapter presents the results of the controller testing and recommends improvements to both the hardware and software designs.

Results

During the testing of the automatic temperature controller, many minor hardware and software errors were corrected. It was hoped that by correcting these errors the controller would work properly. However, this was not the case. In an attempt to determine why the controller did not work properly and to aid in correcting the problems with the controller design, four questions were asked: how well does the microprocessor system perform, how well does each of the interfaces work, how well does the controller change from one temperature to another, and how well does the controller maintain temperature.

Once all the minor errors in hardware and software were corrected, the microprocessor system worked perfectly. The time required for the microprocessor to execute the control algorithm is not as critical as was thought at the beginning of the hardware design. The microprocessor requires 346 milliseconds to determine a heater coil voltage setting once data is available at the A/D converter. This accounts for approximately 60 percent of time required by the A/D to perform a conversion. Therefore, the microprocessor can easily calculate and output a voltage

to the heater coil before the next thermometer sample becomes available. These times, however, are obtained from the algorithm that is actually implemented and not from the more correct algorithm discussed in Chapter IV. Since it is believed that an implementation of the more correct algorithm will require more time, a faster microprocessor system may be needed.

Most of the interfaces between the microprocessor system and the peripheral devices work as they were designed. No problems were found in the heater coil interface, the display interface, or the switch and keypad interface. Unfortunately, there was insufficient time to test the LSI-11 computer interface. Therefore, it is not known whether this interface is functional.

The thermometer interface proved to be the most detrimental to the proper functioning of the automatic temperature controller. While the detection section of this interface does measure the input signals from each of the thermometers, these signals are corrupted with noise by the time they reach the A/D converter. For example, a 1.7 volt signal from the silicon diode contained 400 microvolts of noise. Similar noise levels were noted on the signals from the two control thermometers. It was determined that the most probable source of this noise is Johnson noise generated in the resistors of the detection section. Therefore, some of this noise could be eliminated by using smaller resistor values. Two additional problems were noted in the thermometer interface detection section: output offset voltage, and operational amplifier drift. These two problems result in errors in signal measurement at the A/D converter. These errors were verified by comparing the actual silicon diode voltage with the voltage measured by the automatic temperature

controller. While not specifically measured, these errors are also assumed to be present in the detected signals from the control thermometers. The output offset voltage produces an error of approximately 20 millivolts between the actual silicon diode voltage and the voltage present at the output of the detection section. This problem was not present when the detection section was originally calibrated and it is believed that by recalibrating this section, the offset problem can be alleviated. The second problem, operational amplifier drift, produces a 5 millivolt drift in the measured voltage when the automatic temperature controller is monitoring the silicon diode voltage. This drift appears to be cyclic over a five minute period. The drift error is significant and, therefore, cannot be ignored. The errors caused by the operational amplifier drift were noted especially in one case: when the controller is attempting to control temperature. All of the errors in the thermometer interface -- noise, offset voltage, and amplifier drift -- will have to be corrected in order for the automatic temperature controller to work properly.

Two errors present in the controller software prevented the automatic temperature controller from properly adjusting temperature. These errors were mentioned in the discussion of the control scheme contained in Chapter IV. The first error deals with the equation used to determine the heater coil voltage setting. The equation actually implemented is exactly the same as equation (10) in the last chapter with the exception of the B matrix which is replaced with:

$$B = \begin{bmatrix} x(p-m) & 0 & \dots & 0 & 0 \\ x(p-m+1) & x(p-m) & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x(p-2) & x(p-3) & \dots & x(p-m) & 0 \\ x(p-1) & x(p-2) & \dots & x(p-m+1) & x(p-m) \end{bmatrix}$$

This equation was obtained from an error in the convolution sum equation and, therefore, does not reflect a true measurement of the impulse response of the thermal system. It is not known exactly what impact this error has on the ability of the automatic temperature controller to control temperature. Any error caused by mismeasurement of the impulse response seems to be overshadowed by the second major error in the controller software discussed in the next paragraph.

The second error in the controller software is in how the program handles thermometers whose input signals are monotonically decreasing with temperature. As was stated in Chapter IV, the reciprocal of each input signal sample is used instead of the actual input signal sample. In this way, the signals from these thermometers are made to look monotonically increasing. By using this scheme, however, the controller behaves in a rather bizarre manner. When monitoring the silicon diode voltage, the controller slowly adds voltage across the heater coil when the actual temperature of the sample is much less than the desired temperature. As the temperature of the sample increases towards the desired temperature, the controller continues to add voltage to the heater coil until the desired temperature is overshoot. The explanation for this behavior is as follows: since the reciprocal of the input signals

is used, the controller 'sees' the error between the actual voltage and the desired voltage of the silicon diode as being smallest when the temperature difference is large. As the temperature difference becomes smaller, the controller 'sees' the voltage error as becoming larger and, therefore, increases the heater coil voltage further. Hence, another scheme is needed to account for the thermometers whose signals are monotonically decreasing with temperature. Such a scheme is proposed in the recommendations which follow.

The final question asked how well the automatic temperature controller controls temperature. Once the oscillations of temperature adjustment damp out and the controller switches to one of the two control thermometers, the controller is able to maintain temperature to within 0.1 degrees Kelvin. This deviation is 20 times what was stipulated in the requirements for the automatic temperature controller. This large deviation probably has its source in the noise and drift problems of the thermometer interface detection section. In addition, the reciprocal error discussed in the previous paragraph is also present in the platinum thermometer control since this thermometer produces a current signal which is monotonically decreasing with temperature. By correcting these three sources of error, it is believed that the automatic temperature controller will meet the requirements for temperature control.

Recommendations

During the testing of the automatic temperature controller, many ideas on how the controller could be improved were devised. Contained in the paragraphs below are recommendations for actions to be taken to make the controller functional.

The first recommendation is to measure the impulse response of the thermal system. The lack of this information proved to be a hindrance during the software design. Since the impulse response was not known, the number of points used to represent this response was arbitrarily chosen as five. If, however, the complete impulse response of the thermal system is known, the optimum number of points can be determined.

The second recommendation is to redesign the thermometer interface. While a temporary solution would be to decrease the size of the resistor values in the existing thermometer interface to reduce the Johnson noise, this would not eliminate the operational amplifier drift. To eliminate the drifting problem, compensation networks will probably be required. Some possible methods of reducing noise and amplifier drift are presented in Reference 16.

Implementation of the control equation discussed in Chapter IV is the third recommendation. By using this equation, it is hoped that the controller will have a better measure of the impulse response of the thermal system. This implementation will require a subroutine that can evaluate an $n \times n$ determinant. If this equation is implemented in this way, however, more RAM may be required.

The implementation of equation (10) will require a different method of accounting for thermometers with monotonically decreasing signals. Another method which could be used is to subtract the value of the input signal at 4.2 degrees Kelvin from each input signal sample. While it has not been determined at this time, it is believed that by using this scheme the automatic temperature controller will properly 'see' the errors between desired and actual input signal levels.

The final recommendation is not for the present automated temperature controller, but for future versions of the controller. One of the parameters in the hardware design philosophy stated that the controller should be built from scratch. Now that the present controller is built, however, it is believed that it would be more feasible to build future controllers with pre-built computer boards available on the market. This would eliminate the time spent in constructing and testing of the hardware.

Conclusion

While the present design of the automatic temperature controller is not a complete success, much was learned about the problems associated with temperature control. The hardware and software designs of the present controller form the basis upon which modifications can be made to produce a correctly functioning automatic temperature controller. The steps defined in the recommendations of this chapter outline possible methods by which the automatic temperature controller can be changed to meet the requirements set out in Chapter II.

References

1. Verchot, Edgar A., Jr., Automated Hall Effect Experiment Data Acquisition System (AHEEDAS) Thesis, AFIT/GEO/EE/79D-5, ADAO 80175. Dayton, Ohio: Air Force Institute of Technology, 1979.
2. Putley, E. H., The Hall Effect and Semi-conductor Physics. New York: Dover Publications, Inc., 1960.
3. CGR-1 Technical Data. Westerville, Ohio: Lake Shore Cryotronics, Inc.
4. Technical Data, Series PT100 & PT1000 Platinum RTDs. Westerville, Ohio: Lake Shore Cryotronics, Inc.
5. Calibration Data on Model DT-500P, Serial Number D16865 Silicon Diode Temperature Sensor. Westerville, Ohio: Lake Shore Cryotronics, Inc.
6. White, Guy K., Experimental Techniques in Low Temperature Physics. Oxford: Oxford University Press, 1979.
7. Motorola Microprocessors Data Manual. Motorola, Inc., 1981.
8. Bishop, Ron, Basic Microprocessors and the 6800. Rochelle Park, New Jersey: Hayden Book Company, Inc., 1979.
9. Relay Specifications, Coto Part Number U-20146. Providence, Rhode Island: Coto-coil Co., Inc., 1979.
10. The Linear Control Circuits Data Book. Texas Instruments, Inc., 1976.
11. ICL8068/ICL7104 Pair 16/14/12 Bit Binary A/D Converters for Microprocessors. Intersil, Inc.
12. Operating and Service Manual, Digital Voltage Source, Model 6130C, Binary Logic. Hewlett-Packard, 1973.
13. Memories and Peripherals. Digital Equipment Corporation, 1978.
14. HDSP-6508 16 Segment Solid State Alphanumeric Display. Hewlett-Packard Components, 1978.
15. MCL4490 Hex Contact Bounce Eliminator Data Sheet. Phoenix, Arizona: Motorola Semiconductors.
16. Graeme, Jerald G., Designing with Operational Amplifiers. New York: McGraw-Hill Book Company, 1977.

Appendix A

Controller Board Layout and Pin Connections

This appendix contains the chip layout and pin connections for the two boards contained in the automatic temperature controller. Complete details on the hardware system can be obtained from Dr. Patrick M.

Hemenger, AFWAL/MLPO, Wright-Patterson AFB, Ohio 45433.

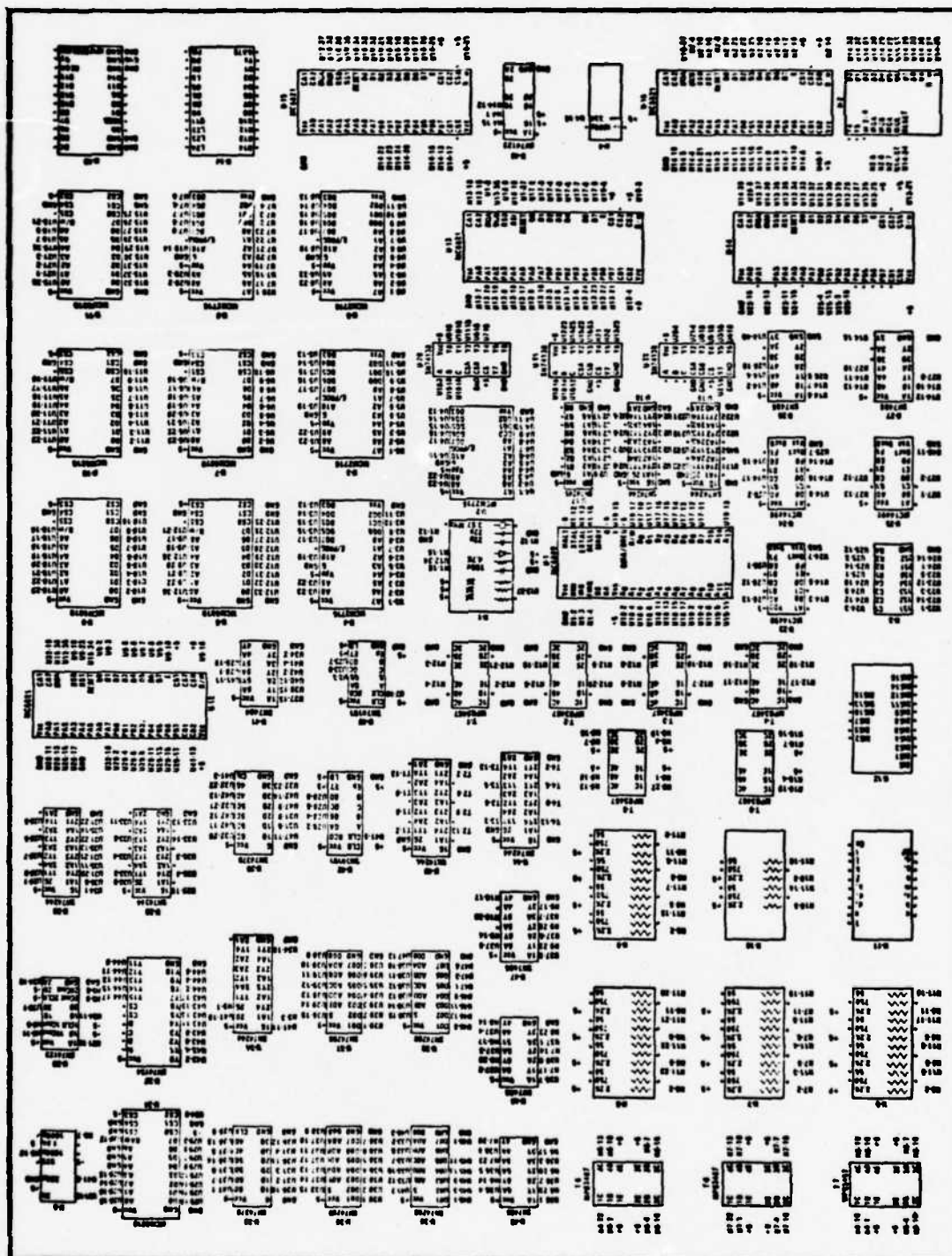


Figure A-1. Microcomputer Board Layout and Connections

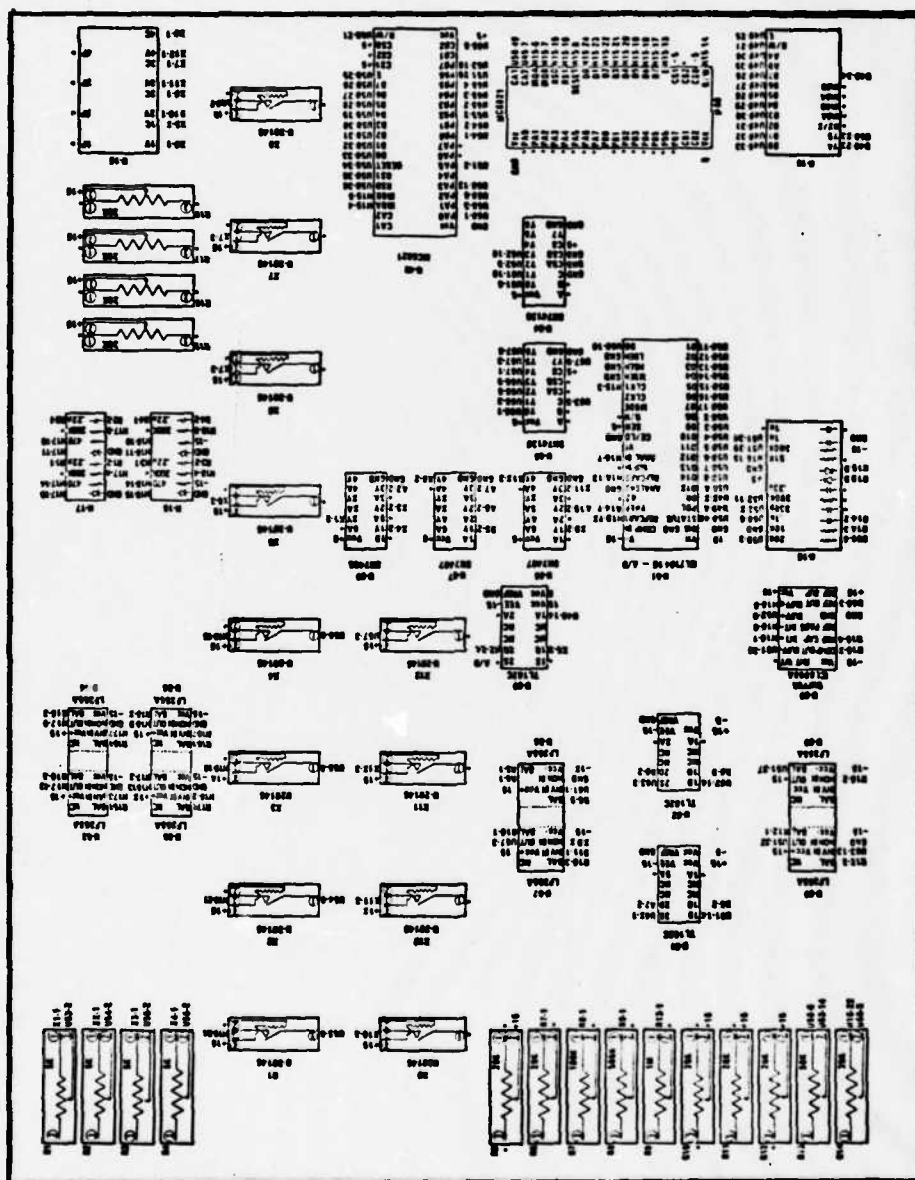


Figure A-2. Thermometer Interface Board Layout and Pin Connections

Appendix B

Controller Program Assembly Listing

This appendix contains the assembly listing of the software contained in the automatic temperature controller. To aid in understanding the code the assembly listing is supplemented with a pseudo-PASCAL program listing.

NAM CNTRLR

★

★ THIS IS THE AUTOMATIC TEMPERATURE CONTROLLER SOFTWARE CODE. INCLUDED ★
★ WITH THE ASSEMBLY MNEMONICS IS A PSEUDO-PASCAL LISTING. ★

★

★ MEMORY REQUIREMENTS: ★

★

★ ROM 10K BYTES ★

★ RAM 640 BYTES ★

★

★

★ DEFINE PIA DATA PORT LOCATIONS AND CONTROL REGISTER LOCATIONS ★

★

DISPAD EQU \$1000

DISPDA EQU \$1001

SWITCH EQU \$1002

KEYBRD EQU \$1003

PSFLGS EQU \$1005

PSDAT0 EQU \$1006

PSDAT1 EQU \$1007

SENFLG EQU \$1008

SENSEL EQU \$1009

SENDA0 EQU \$100A

SENDA1 EQU \$100B

CONDA0 EQU \$100C

CONDA1 EQU \$100D

DISPC0 EQU \$1010

DISPC1 EQU \$1011

SWCHCR EQU \$1012

KEYCR EQU \$1013

PSCRO EQU \$1015

PSCR1 EQU \$1016

PSCR2 EQU \$1017

SENCRO EQU \$1018

SENCR1 EQU \$1019

SENCR2 EQU \$101A

SENCR3 EQU \$101B

COMCRO EQU \$101C

COMCR1 EQU \$101D

*
* DEFINE ORIGIN LOCATIONS, STACK STARTING LOCATION, AND OTHER MISCELA- *
* NEOUS EQUATES. *
*

DATA EQU \$0000
PROG EQU \$D800
SUB EQU \$F000
VECTOR EQU \$FFF2
USTART EQU \$017F
SSTART EQU \$027F
EOT EQU \$FF

```

*****
*
*   DEFINE RAM DATA LOCATIONS.
*
*****

```

```

      ORG DATA
DIG1  RMB 16
ROTARY RMB 1
TOGGLE RMB 1
PRIMRY RMB 1
SECDRY RMB 1
PSCURR RMB 4
LDRES  RMB 4
PSMAX  RMB 4
ERROR  RMB 4
NUMSEN RMB 1
SEN1   RMB 17
SEN2   RMB 17
SEN3   RMB 17
SEN4   RMB 17
YD     RMB 4
YS     RMB 4
Y0     RMB 4
Y1     RMB 4
Y2     RMB 4
Y3     RMB 4
Y4     RMB 4
U0     RMB 4
U1     RMB 4
U2     RMB 4
U3     RMB 4
U4     RMB 4
U5     RMB 4

```

ORG PROG
IRQVEC JMP IRQ
NMIVEC JMP NMI
SWIVEC JMP SWI
RSTVEC JMP RESET

* INTERRUPT VECTOR JUMPS

```

*****
*
*   THIS IS THE FAST INTERRUPT ROUTINE.  IT CAN BE USED TO CHECK THE
*   INTEGRITY OF THE ARITHMETIC SUBROUTINES.
*
*****

```

```

FIRQ   LDS #SSTART      * RESET STACKS
       LDU #USTART

       * REPEAT
PNTF00 JSR CRLF          *   CRLF
       JSR OUTSTR        *   WRITE('FUNCTION(0-9)?')
       FCC /FUNCT/
       FCC /ION(0/
       FCC /-9)?/
       FCB EOT
       JSR RDDIG          *   RDDIG
       BVS PNTF00         *   UNTIL V CLEAR
       TSTA              *   CASE A OF
       BNE PNTF01
       JSR RDKEY          *   0(RDKEY):  RDKEY
       BRA PNTF02

PNTF01 DECA
       BNE PNTF03
       JSR PLUS           *   1(ADD):   PLUS
       BRA PNTF02

PNTF03 DECA
       BNE PNTF04
       JSR NEGATE         *   2(SUBTRACT): NEGATE
       BRA PNTF02         *           PLUS

PNTF04 DECA
       BNE PNTF05
       JSR MLTPLY         *   3(MULTIPLY): MLTPLY
       BRA PNTF02

PNTF05 DECA
       BNE PNTF06
       JSR INVRSE         *   4(DIVIDE):  INVRSE
       JSR MLTPLY         *           MLTPLY
       BRA PNTF02

PNTF06 DECA
       BNE PNTF07
       JSR INVRSE         *   5(1/X):   INVRSE
       BRA PNTF02

PNTF07 DECA
       BNE PNTF08
       JSR NEGATE         *   6(CHANGE SIGN): NEGATE
       BRA PNTF02

PNTF08 DECA
       BNE PNTF09
       JSR XINCGY         *   7(X INTERCHANGE Y): XINCGY
       BRA PNTF02

PNTF09 DECA
       BNE PNTFOA
       LEAU 4,U           *   8(STACK ROLLDOWN): REMOVE X

```

BRA PNTF02
PNTFOA JSR COPY
PNTF02 JSR COPY
JSR CRFL
JSR OUTDEC

PNTFOB JSR INCH
CMPA #'R
BNE PNTFOB
BRA PNTF00

* 9(COPY): COPY
* COPY
* CRLF
* OUTPUT DECIMAL NUMBER
* REPEAT
* INCH
* UNTIL A = 'R'

```

*****
*
*   THIS IS THE INTERRUPT REQUEST HANDLER.  INTERRUPT PRIORITY IS AS
*   FOLLOWS: LSI-11 COMPUTER, KEYPAD, ROTARY AND TOGGLE SWITCHES, AND
*   THE HP-6130C POWER SUPPLY.
*
*****

```

```

IRQ   LDA CONCR0      * IF COMPUTER INTERRUPT
      ANDA #$81
      CMPA #$81
      BNE PNT100

      JMP COMPINT      * THEN
                       * GO TO COMINT
PNT100 LDA KEYCR      * IF KEYBOARD INTERRUPT
      ANDA #$81
      CMPA #$81
      BNE PNT101

      JMP KEYINT      * THEN
                       * GO TO KEYINT
PNT101 LDA SWCHCR     * IF SWITCH INTERRUPT
      BMI PNT102
      LSLA
      BPL PNT103

      JSR RDSWCH      * THEN
                       * RDSWCH
      RTI             * RETURN
PNT103 LDA PSCRO      * IF NOT POWER SUPPLY INTERRUPT
      BMI PNT104

      RTI             * THEN
                       * RETURN
                       * REPEAT
PNT104 JSR CRLF      * CRLF
      JSR SCRSTR      * WRITE('POWER SUPPLY CURRENT LIMIT EXCEEDED.
      FCC /POWER/      * SHOULD THIS LIMIT BE INCREASED?')
      FCC / SUPP/
      FCC /LY CU/
      FCC /RRENT/
      FCC / LIM1/
      FCC /T EXC/
      FCC /EDED./
      FCC / SH/
      FCC /OULD /
      FCC /THIS /
      FCC /LIMIT/
      FCC / BE 1/
      FCC /NCREA/
      FCC /SED?/
      FCB EOT
      JSR RDANSW      * RDANSW
      BVS PNT104      * UNTIL V CLEAR
      CMPA #'Y'      * IF A <> 'Y'
      BEQ PNT105

      * THEN

```


	JMP BYE	* GO TO BYE
PNT105	JSR CRLF	* CRLF
	JSR SCRSTR	* WRITE('PRESENT POWER SUPPLY OUTPUT CURRENT
	FCC /PRESE/	* LIMIT = ')
	FCC /NT PO/	
	FCC /WER S/	
	FCC /UPPLY/	
	FCC / OUTP/	
	FCC /UT CU/	
	FCC /KRENT/	
	FCC / LIMI/	
	FCC /T = /	
	FCB EOT	
	LDX #PSCURR	* WRITE(PSCURR)
	JSR RECALL	
	JSR SCRDEC	
	JSR SCRSTR	
	FCC / /	
	FCC / /	
	FCC / /	
	FCB EOT	
	JSR SETCUR	* SETCUR
	LDX #LDRES	* PSMAX := PSCURR * LDRES
	JSR RECALL	
	JSR MLTPY	
	LDX #PSMAX	
	JSR STORE	
	LEAU 4,U	
	RTI	* RETURN

```

*****
*
*   THIS IS THE SOFTWARE INTERRUPT HANDLER.  A SOFTWARE INTERRUPT IS GEN-
*   ERATED BY THE CALLING PROGRAM FOR THE FOLLOWING REASONS:  INVALID
*   INPUT FROM THE TOGGLE SWITCH, UNDEFINED COMPUTER INPUT, AN ATTEMPT
*   AT DIVISION BY ZERO, OR INVALID OR OVERLOAD STATUS OF THE A/D CON-
*   VERTER.  EACH SWI HAS A DEFAULT VALUE WHICH IS IMHERENT IN THE CALLING
*   PROGRAM.  IF THE DEFAULT IS NOT TAKEN, THE PROGRAM IS TERMINATED.
*
*****

```

```

SWI   LDA [10,S]      * READ(A)
      INC 11,S        * CORRECT RETURN ADDRESS
      BNE PNT200
      INC 10,S

PNT200 TSTA           * CASE A OF
      BNE PNT201

      * 0: REPEAT
      *      CRLF
      *      WRITE('HARDWARE FAILURE . . . DEFAULT?')

PNT202 JSR CRLF
      JSR SCRSTR
      FCC /HARDW/
      FCC /ARE F/
      FCC /AILUR/
      FCC /E . ./
      FCC / . DE/
      FCC /FAULT/
      FCC /?/
      FCB EOT
      JSR RDANSW      *      RDANSW
      BVS PNT202      *      UNTIL V CLEAR
      BRA PNT203

PNT201 DEC A
      BNE PNT204

      * 1: REPEAT
      *      CRLF
      *      WRITE('UNDEFINED COMPUTER INPUT . . .
      *                                     DEFAULT?')

      FCC /UNDEF/
      FCC /INED /
      FCC /COMPU/
      FCC /TER I/
      FCC /NPUT /
      FCC / . . ./
      FCC / DEFA/
      FCC /ULT?/
      FCB EOT
      JSR RDANSW      *      RDANSW
      BVS PNT205      *      UNTIL V CLEAR
      BRA PNT203

PNT204 DEC A
      BNE PNT206

      * 2: REPEAT
      *      CRLF
      *      WRITE('DIVISION BY ZERO . . . DEFAULT?')

PNT207 JSR CRLF
      JSR SCRSTR

```

```

FCC /DIVIS/
FCC /ION B/
FCC /Y ZER/
FCC /O . ./
FCC / . DE/
FCC /FAULT/
FCC /?/
FCB EOT
JSR RDANSW      *      RDANSW
BVS PNT207      *      UNTIL V CLEAR
BRA PNT203
PNT206 DEC A
BNE PNT208

*      3: REPEAT
*      CRLF
*      WRITE('A/D DATA MISSED . . . DEFAULT?')

PNT209 JSR CRLF
JSR SCRSTR
FCC /A/D D/
FCC /ATA M/
FCC /ISSED/
FCC / . . /
FCC / . DEF/
FCC /AULT?/
FCB EOT
JSR RDANSW      *      RDANSW
BVS PNT209      *      UNTIL V CLEAR
BRA PNT203

*      4: REPEAT
*      CRLF
*      WRITE('A/D OVERLOAD . . . DEFAULT?')

PNT208 JSR CRLF
JSR SCRSTR
FCC /A/D O/
FCC /VERLO/
FCC /AD . /
FCC / . . D/
FCC /EFAUL/
FCC /T?/
FCB EOT
JSR RDANSW      *      RDANSW
BVS PNT208      *      UNTIL V CLEAR
PNT203 CHPA #'Y' *      IF A = 'Y'
BNE PNT208

*      THEN
*      RETURN
*      ELSE
*      GO TO BYE

RTI
PNT20B JMP BYE

```

```

*****
*
*   THIS IS THE NON MASKABLE INTERRUPT ROUTINE.  IT IS CAPABLE OF DIS-
*   PLAYING THE CONTENTS OF EACH OF THE MPU REGISTERS AS THEY EXISTED AT
*   THE TIME OF THE INTERRUPT.  IN ADDITION, ANY MEMORY LOCATION MAY BE
*   EXAMINED USING THIS ROUTINE.  THIS ROUTINE, ALONG WITH AN EXTERNAL
*   WORD RECOGNIZER, CAN BE USED TO SINGLE-STEP THROUGH THE PROGRAM.
*
*****

```

```

NMI      JSR CRLF      * CRLF
                        * REPEAT
PNT500   JSR INCH      * INCH
          PSHS A
          JSR CRLF      * CRLF
          LDA ,S+        * CASE A OF
          BNE PNT501
          JSR OUTSTR     * 0: WRITE('ADDR ')
          FCC /ADDR /
          FCB EOT
          LEAS -2,S
          JSR INHEX      * INHEX
          JSR OUTCH      * OUTCH
          LSLA
          LSLA
          LSLA
          LSLA
          STA ,S
          BSR INHEX      * INHEX
          JSR OUTCH      * OUTCH
          ORA ,S
          STA ,S
          BSR INHEX      * INHEX
          JSR OUTCH      * OUTCH
          LSLA
          LSLA
          LSLA
          LSLA
          STA 1,S
          BSR INHEX      * INHEX
          JSR OUTCH      * OUTCH
          ORA 1,S
          STA 1,S
          JSR OUTS
          LDA [,S++]     * A := [ADDR]
          JSR OUTHEX     * OUTHEX
          BRA PNT500
PNT501   DECA
          BNE PNT502
          JSR OUTSTR     * 1: WRITE('CC ')
          FCC /CC /
          FCB EOT
          LDA ,S         * WRITE(CC)
          JSR OUTHEX

```

```

        BRA PNT500
PNT502 DECA
        BNE PNT503
        JSR OUTSTR      *      2:  WRITE('A  ')
        FCC /A  /
        FCB EOT
        LDA 1,S          *      WRITE(A)
        JSR OUTHEX
        BRA PNT500
PNT503 DECA
        BNE PNT504
        JSR OUTSTR      *      3:  WRITE('B  ')
        FCC /B  /
        FCB EOT
        LDA 2,S          *      WRITE(B)
        JSR OUTHEX
        BRA PNT500
PNT504 DECA
        BNE PNT505
        JSR OUTSTR      *      4:  WRITE('DP  ')
        FCC /DP  /
        FCB EOT
        LDA 3,S          *      WRITE(DP)
        JSR OUTHEX
        BRA PNT500
PNT505 DECA
        BNE PNT506
        JSR OUTSTR      *      5:  WRITE('X  ')
        FCC /X  /
        FCB EOT
        LDA 4,S          *      WRITE(X)
        JSR OUTHEX
        LDA 5,S
        JSR OUTHEX
        BRA PNT500
PNT506 DECA
        BNE PNT507
        JSR OUTSTR      *      6:  WRITE('Y  ')
        FCC /Y  /
        FCB EOT
        LDA 6,S          *      WRITE(Y)
        JSR OUTHEX
        LDA 7,S
        JSR OUTHEX
        BRA PNT500
PNT507 DECA
        BNE PNT508
        JSR OUTSTR      *      7:  WRITE('US  ')
        FCC /US  /
        FCB EOT
        LDA 8,S          *      WRITE(US)
        JSR OUTHEX
        LDA 9,S
        JSR OUTHEX

```

	BRA PNT500		
PNT508	DECA		
	BNE PNT509	*	8: WRITE('SP')
	JSR OUTSTR		
	FCC /SP /		
	FCB EOT		
	TFR S,D	*	WRITE(SP)
	ADDD #SOC		
	PSHS B		
	JSR OUTHEX		
	PULS A		
	JSR OUTHEX		
	BRA PNT500		
PNT509	DECA		
	BNE PNT50A	*	9: WRITE('PC')
	JSR OUTSTR		
	FCC /PC /		
	FCB EOT		
	LDA 10,S	*	WRITE(PC)
	JSR OUTHEX		
	LDA 11,S		
	JSR OUTHEX		
	BRA PNT500		
PNT50A	CMPA #S03	.	
	BNE PNT500	*	C: RETURN
	RTI	*	FOREVER
INHEX	JSR INCH	*	INCH
	CMPA #S09	*	IF A > 9
	BLS PNT510		
		*	THEN
	CMPS #'. .	*	IF A = '. .'
	BNE PNT511		
		*	THEN
	LDA #'A	*	A := 'A'
	BRA PNT510		
		*	ELSE
PNT511	CMPS #'I	*	IF A = 'I'
	BNE PNT512		
		*	THEN
	LDA #'B	*	A := 'B'
	BRA PNT510		
		*	ELSE
PNT512	CMPS #'Y	*	IF A = 'Y'
	BNE PNT513		
		*	THEN
	LDA #'D	*	A := 'D'
	BRA PNT510		
		*	ELSE
PNT513	CMPS #'N	*	IF A = 'N'
	BNE PNT514		
		*	THEN
	LDA #'E	*	A := 'E'
	BRA PNT510		

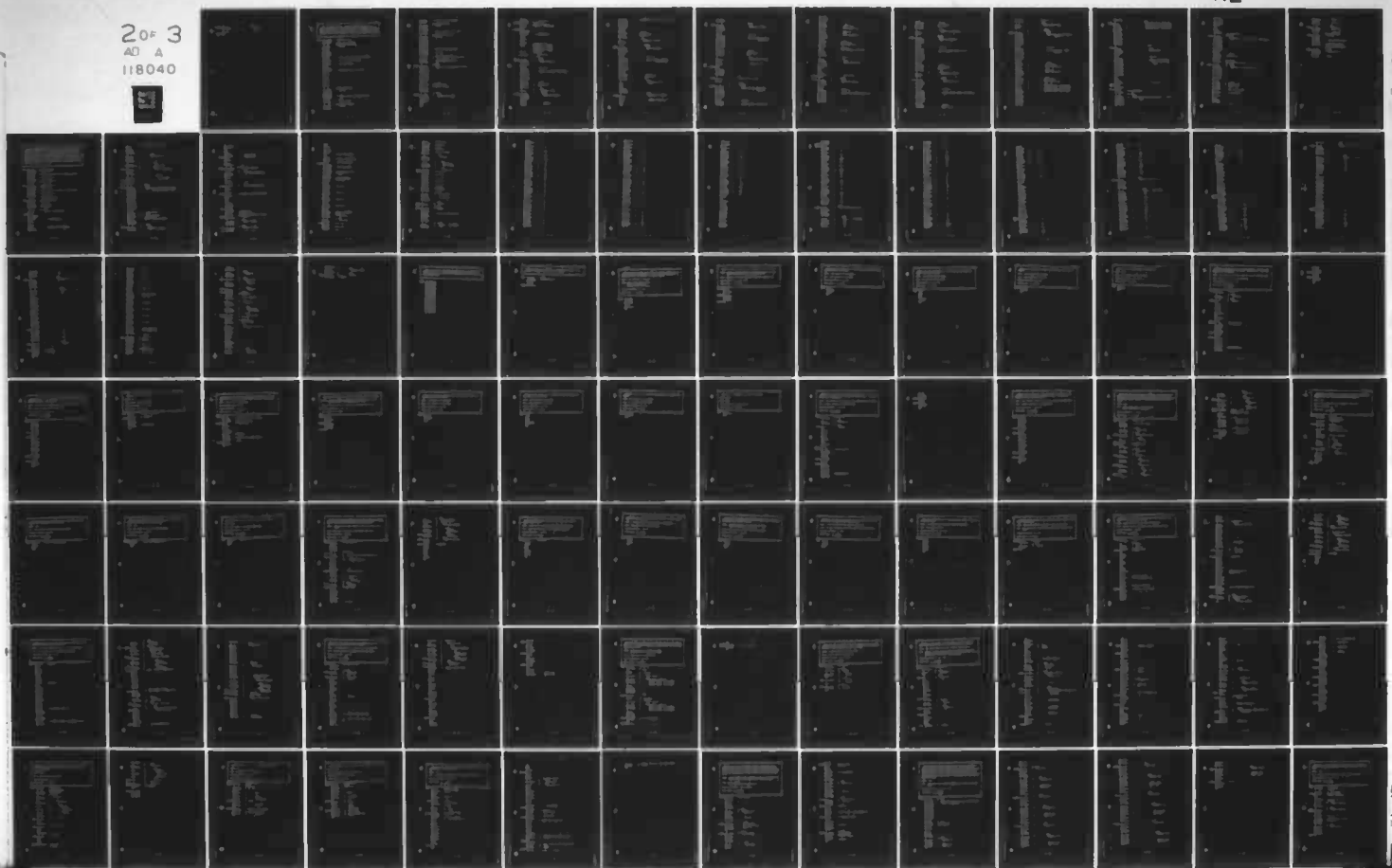
AD-A118 040

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 13/1
AN AUTOMATED TEMPERATURE CONTROLLER FOR THE ADVANCED HALL EFFEC--ETC(1)
MAR 82 D J PAGE
AFIT/GE0/EE/82M-1

UNCLASSIFIED

NL

2 of 3
AD A
118040



0 A
804

PNT514 CMPA #'R
BNE PNT510

LDA #'F
PNT510 RTS

ELSE
IF A = 'R'
THEN
A := 'F'

```

*****
*
*   THIS IS THE RESET ROUTINE.  UPON POWER-UP OR AN MPU RESET, THIS
*   ROUTINE INITIALIZES ALL MPU REGISTERS, PIA PORTS, AND PERIPHERAL EQUIP-
*   MENT.  THIS ROUTINE ALSO PROMPTS THE USER FOR ALL INFORMATION ABOUT THE
*   EXPERIMENT NECESSARY FOR TEMPERATURE CONTROL.  A LIMITED AMOUNT OF ERROR
*   CHECKING IS DONE BY THIS ROUTINE TO INSURE THAT THE INPUT IS REASONABLE.
*
*****

```

```

RESET  LDS #SSTART      * INITIALIZE STACKS
        LDU #USTART
        ANDCC #SBF      * UNMASK FAST INTERRUPT
                        * INITIALIZE POWER SUPPLY
        CLR PSCRO       *   SET DDR
        CLR PSCR1
        CLR PSCR2
        LDA #SOF
        STA PSFLGS
        LDD #SFFFF
        STD PSDAT0
        LDA #S05        *   ENABLE POWER SUPPLY OVERLOAD INTERRUPT
        STA PSCRO
        LDD #S042C      *   ENABLE POWER SUPPLY UPDATA
        STD PSCR1
        LDD #S0000      *   ZERO POWER SUPPLY
        PSHU D
        PSHU D
        JSR SETPWR
        LDD #S0000      *   INITIALIZE POWER SUPPLY HISTORY
        STD U1
        STD U1+2
        STD U2
        STD U2+2
        STD U3
        STD U3+2
        STD U4
        STD U4+2
        STD U5
        STD U5+2

        CLR DISPC0      * INITIALIZE DISPLAY
        CLR DISPC1      *   SET DDR
        LDD #S0FFF
        STD DISPAD
        LDD #S3C2C      *   DISABLE DISPLAY, SET DISPLAY UPDATE
        STD DISPC0
        JSR CRLF        *   CRLF
        LDA #S34        *   ENABLE DISPLAY
        STA DISPC0

        * INITIALIZE KEYBOARD
        CLR KEYCR       *   SET DDR
        LDA #S00
        STA KEYBRD

```

LDA #06	* SET EDGE, DISABLE INTERRUPT
STA KEYCR	
	* INITIALIZE COMPUTER
CLR COMCRO	* SET DDR
CLR COMCR1	
LDD #0000	
STD COMDAO	
LDD #043C	* DISABLE INTERRUPT, SIGNAL INVALID
STD COMCRO	
	* INITIALIZE SENSORS, A/D
CLR SENCRO	* SET DDR
CLR SENCRI	
CLR SENCRI2	
CLR SENCRI3	
LDD #0FFF	
STD SENFLG	
LDD #0000	
STD SENDAO	
LDD #043C	* DISABLE SENSOR SELECT(CNTL)
STD SENCRO	
LDD #0404	* SET EDGE
STD SENCRI2	
CLR SENFLG	* TURN SENSOR POWER OFF
LDA #0A0	* HOLD A/D, DISABLE A/D INPUT, DISABLE SENSOR SELECT,
STA SENSEL	* SET GAIN = 1
LDA #34	* ENABLE SENSOR SELECT(CNTL)
STA SENCRI	
	* REPEAT
PNT300 JSR CRLF	* CRLF
JSR SCRSTR	* WRITE('HOW MANY SENSORS WILL BE USED(1,2,3,4)?')
FCC /HOW M/	
FCC /ANY S/	
FCC /ENSOR/	
FCC /S WIL/	
FCC /L BE /	
FCC /USED(/	
FCC /1,2,3/	
FCC / ,4)?/	
FCB EOT	
JSR RDDIC	* RDDIC
BVS PNT300	* UNTIL V CLEAR
TSTA	* IF (A=0) OR (A>4)
BEQ PNT301	
CMPA #04	
BLS PNT302	
	* THEN
PNT301 JSR CRLF	* CRLF
JSR SCRSTR	* WRITE('INVALID INPUT')
FCC /INVAL/	
FCC /ID IN/	
FCC /PUT /	
FCC / /	
FCC / /	
FCC / /	

```

FCB EOT
BRA PNT300
PNT302 STA NUMSEN
CLR ,S
LDA #S01
PSHS A
PNT303 LDA ,S
CMPA NUMSEN
BHI PNT304
LDY #SEN1
LDA ,S
DEC A
LDB #S11
MUL
LEAY B,Y
LDA ,S
STA ,Y

LDA 1,S
BNE PNT305

PNT306 JSR CRLF
JSR SCRSTR
FCC /IS SE/
FCC /NSOR /
FCB EOT
LDA ,S
JSR SCROCH
JSR SCRSTR
FCC / THE /
FCC /PRIMA/
FCC /RY SE/
FCC /NSOR(/
FCC /Y,N)?/
FCB EOT
JSR RDANSW
BVS PNT306
CMPA #'Y
BNE PNT305

DEC 1,S

PNT307 JSR CRLF
JSR SCRSTR
FCC /DOES /
FCC /SENSO/
FCC /R /
FCB EOT
LDA ,S
JSR SCROCH

```

```

* GO TO PNT300
* NUMSEN := A
* PRIMFND := FALSE
* FOR I := 1 TO NUMSEN DO

* Y := ADDR(SEN(I))

* SEN(I).NUM := I
* SEN(I).POL := POSITIVE
* SEN(I).SIG := VOLTAGE
* SEN(I).PWRREQD := FALSE
* SEN(I).TYPE := PRIMARY
* IF PRIMFND
* THEN
* GO TO PNT305
* ELSE
* REPEAT
* CRLF
* WRITE('IS SENSOR ')

* WRITE(1)
* WRITE(' THE PRIMARY SENSOR(Y,N)?')

* RDANSW
* UNTIL V CLEAR
* IF A = 'Y'

* THEN
* PRIMFND := TRUE
* REPEAT
* CRLF
* WRITE('DOES SENSOR ')

* WRITE(1)

```

	JSR SCRSTR	*	WRITE(' HAVE A POSITIVE SLOPE(Y,N)?')
	FCC / HAVE/		
	FCC / A PO/		
	FCC /SITIV/		
	FCC /E SLO/		
	FCC /PE(Y,/		
	FCC /N)?/		
	FCB EOT		
	JSR RDANSW	*	RDANSW
	BVS PNT307	*	UNTIL V CLEAR
	CMPA #'N	*	IF A = 'N'
	BNE PNT308		
		*	THEN
	LDA ,Y	*	SEN(I).POL := NEGATIVE
	ORA #\$80		
	STA ,Y		
		*	REPEAT
PNT308	JSR CRLF	*	CRLF
	JSR SCRSTR	*	WRITE('DOES SENSOR ')
	FCC /DOES /		
	FCC /SENSO/		
	FCC /R /		
	FCB EOT		
	LDA ,S	*	WRITE(I)
	JSR SCROCH	*	
	JSR SCRSTR	*	WRITE(' PRODUCE A VOLTAGE
	FCC / PROD/	*	SIGNAL(Y,N)?')
	FCC /UCE A/		
	FCC / VOLT/		
	FCC /AGE S/		
	FCC /IGNAL/		
	FCC /(Y,N)/		
	FCC /?/		
	FCB EOT		
	JSR RDANSW	*	RDANSW
	BVS PNT308	*	UNTIL V CLEAR
	CMPA #'Y	*	IF A = 'Y'
	BNE PNT309		
		*	THEN
	LDA ,S	*	PRIMARY := VOLT:I
	DECA		
	LSLA		
	LSLA		
	ORA #\$30		
	STA PRIMRY		
	BRA PNT30A		
		*	ELSE
PNT309	LDA ,Y	*	SEN(I).SIG := CURRENT
	ORA #\$40		
	STA ,Y		
	LDA ,S	*	PRIMARY := CURRENT:I
	DECA		
	LSLA		
	LSLA		

ORA #S90		
STA PRIMRY		
	*	
PNT30B JSR CRLF	*	REPEAT
JSR SCRSTR	*	CRLF
FCC /DOES /		WRITE('DOES SENSOR ')
FCC /SENSO/		
FCC /R /		
FCB EOT		
LDA ,S	*	WRITE(I)
JSR SCROCH		
JSR SCRSTR	*	WRITE(' REQUIRE INTERNAL
FCC / REQU/	*	POWER(Y,N)?')
FCC /IRE I/		
FCC /NTERN/		
FCC /AL PO/		
FCC /WER(Y/		
FCC / ,N)?/		
FCB EOT		
JSR RDANSW	*	RDANSW
BVS PNT30B	*	UNTIL V CLEAR
CMPA #'Y	*	IF A = 'Y'
BNE PNT30A		
	*	THEN
LDA ,Y	*	SEN(I).PWRRQD := TRUE
ORA #S20		
STA ,Y		
LDA #S10	*	SENPR(I) := ON
LDB ,S		
PNT30C LSRA		
DECB		
BNE PNT30C		
ORA SENDAO		
STA SENDAO		
BRA PNT30A		
	*	ELSE
PNT305 LDA ,Y	*	SEN(I).TYPE := SECONDARY
ORA #S10		
STA ,Y		
	*	REPEAT
PNT30D JSR CRLF	*	REPEAT
JSR SCRSTR	*	CRLF
FCC /OVER /	*	WRITE('OVER WHAT RANGE SHOULD
FCC /WHAT /	*	SENSOR ')
FCC /RANGE/		
FCC / SHOU/		
FCC /LD SE/		
FCC /NSOR /		
FCB EOT		
LDA ,S	*	WRITE(I)
JSR SCROCH		
JSR SCRSTR	*	WRITE(' BE USED? START POINT?')
FCC / BE U/		

FCC /SED? /		
FCC / STA/		
FCC /RT PO/		
FCC /INT?/		
FCB EOT		
JSR RDKEY	*	RDKEY
BVS PNT30D	*	UNTIL V CLEAR
LEAX 1,Y	*	SEN(I).URNG1 := [U]
JSR STORE		
LEAU 4,U	*	REMOVE [U]
JSR CRLF	*	CRLF
JSR OUTSTR	*	WRITE('END POINT?')
FCC /END P/		
FCC /OINT?/		
FCB EOT		
JSR RDKEY	*	RDKEY
BVS PNT30D	*	UNTIL V CLEAR
LEAX 5,Y	*	SEN(I).URNG2 := [U]
JSR STORE		
LEAU 4,U	*	REMOVE [U]
PSHS Y	*	IF SEN(I).URNG1 = SEN(I).URNG2
LEAX 1,Y		
LEAY 5,Y		
JSR CMPXY		
BNE PNO30E	.	
	*	THEN
PULS Y		
JSR CRLF	*	CRLF
JSR SCRSTR	*	WRITE('INVALID RANGE')
FCC /INVAL/		
FCC /ID RA/		
FCC /NGE /		
FCC / /		
FCC / /		
FCC / /		
FCB EOT		
BRA PNT30D	*	GO TO PNT30D
PNO30E PULS Y	*	
	*	REPEAT
PNT30E JSR CRLF	*	CRLF
JSR SCRSTR	*	WRITE('DOES SENSOR')
FCC /DOES /		
FCC /SENSO/		
FCC /R /		
FCB EOT		
LDA ,S	*	WRITE(I)
JSR SCROCH		
JSR SCRSTR	*	WRITE(' HAVE A POSITIVE SLOPE(Y,N)?')
FCC / HAVE/		
FCC / A PO/		
FCC /SITIV/		
FCC /E SLO/		
FCC /PE(Y,/		
FCC /N)?/		

FCB EOT			
JSR RDANSW	*	RDANSW	
BVS PNT30E	*	UNTIL V CLEAR	
CMP A #'N	*	IF A = 'N'	
BNE PNT30F			
	*	THEN	
LDA ,Y	*	SEN(1).POL := NEGATIVE	
ORA #\$80			
STA ,Y			
	*	REPEAT	
PNT30F JSR CRLF	*	CRLF	
JSR SCRSTR	*	WRITE('DOES SENSOR ')	
FCC /DOES /			
FCC /SENSO/			
FCC /R /			
FCB EOT			
LDA ,S	*	WRITE(1)	
JSR SCROCH			
JSR SCRSTR	*	WRITE(' PRODUCE A VOLTAGE SIGNAL(Y,N)?')	
FCC / PROD/			
FCC /UCE A/			
FCC / VOLT/			
FCC /AGE S/			
FCC /IGNAL/			
FCC /(Y,N)/			
FCC /?/			
FCB EOT			
JSR RDANSW	*	RDANSW	
BVS PNT30F	*	UNTIL V CLEAR	
CMP A #'N	*	IF A = 'N'	
BNE PNT30A			
	*	THEN	
LDA ,Y	*	SEN(1).SIG := CURRENT	
ORA #\$40			
STA ,Y			
	*	REPEAT	
PNT310 JSR CRLF	*	CRLF	
JSR SCRSTR	*	WRITE('DOES SENSOR ')	
FCC /DOES /			
FCC /SENSO/			
FCC /R /			
FCB EOT			
LDA ,S	*	WRITE(1)	
JSR SCROCH			
JSR SCRSTR	*	WRITE(' REQUIRE INTERNAL POWER(Y,N)?')	
FCC / REQU/			
FCC /IRE I/			
FCC /NTERN/			
FCC /AL PO/			
FCC /WER(Y/			
FCC /,N)?/			
FCB EOT			
JSR RDANSW	*	RDANSW	
BVS PNT310	*	UNTIL V CLEAR	

CMP A #'Y	*	IF A = 'Y'
BNE PNT30A	*	
	*	THEN
LDA ,Y	*	SEN(1).PWRRQD := TRUE
ORA #S20		
STA ,Y	*	
	*	REPEAT
	*	REPEAT
PNT311 JSR CRLF	*	CRLF
JSR SCRSTR	*	WRITE('OVER WHAT RANGE SHOULD
FCC /OVER /	*	SENSOR ')
FCC /WHAT /		
FCC /RANGE/		
FCC / SHOU/		
FCC /LD SE/		
FCC /NSOR /		
FCB EOT		
LDA ,S	*	WRITE(1)
JSR SCROCH		
JSR SCRSTR	*	WRITE(' BE POWERED? START
FCC / BE P/	*	POINT?')
FCC /OWERE/		
FCC /D? /		
FCC /START/		
FCC / POIN/		
FCC /T?/		
FCB EOT		
JSR RDKEY	*	RDKEY
BVS PNT311	*	UNTIL V CLEAR
LEAX 9,Y	*	SEN(1).PRNG1 := [U]
JSR STORE		
LEAU 4,U	*	REMOVE [U]
JSR OUTSTR	*	WRITE('END POINT?')
FCC /END P/		
FCC /OINT?/		
FCB EOT		
JSR RDKEY	*	RDKEY
BVS PNT311	*	UNTIL V CLEAR
LEAX 13,Y	*	SEN(1).PRNG2 := [U]
JSR STORE		
LEAU 4,U	*	REMOVE [U]
PSHS Y	*	IF (SEN(1).PRNG1 > SEN(1).URNG1)
LEAX 9,Y	*	OR (SEN(1).PRNG1 > SEN(1).URNG1)
LEAY 1,Y	*	OR (SEN(1).PRNG2 < SEN(1).URNG2)
JSR CMPXY	*	OR (SEN(1).PRNG2 < SEN(1).URNG1)
BHI PTO304		
LEAY 4,Y		
JSR CMPXY		
BHI PTO304		
LEAX 4,X		
JSR CMPXY		
BLO PTO304		
LEAY -4,Y		
JSR CMPXY		

BHS PT0305	*	THEN
PT0304 LDY ,S	*	IF (SEN(I).PRNG1 < SEN(I).URNG1)
LEAX 9,Y	*	OR (SEN(I).PRNG1 < SEN(I).URNG2)
LEAY 1,Y	*	OR (SEN(I).PRNG2 > SEN(I).URNG2)
JSR CMPXY	*	OR (SEN(I).PRNG2 > SEN(I).URNG1)
BLO PT0306		
LEAY 4,Y		
JSR CMPXY		
BLO PT0306		
LEAX 4,X		
JSR CMPXY		
BHI PT0306		
LEAY -4,Y		
JSR CMPXY		
BLS PT0305		
	*	THEN
PT0306 PULS Y		
JSR CRLF	*	CRLF
JSR SCRSTR	*	WRITE('SENSOR ')
FCC /SENSO/		
FCC /R /		
FCB EOT		
LDA ,S	*	WRITE(I)
JSR SCROCH		
JSR SCRSTR	*	WRITE(' MUST BE POWERED
FCC / MUST/	*	OVER THE ENTIRE USABLE
FCC / BE P/	*	RANGE ')
FCC /OWERE/		
FCC /D OVE/		
FCC /R THE/		
FCC / ENTI/		
FCC /RE US/		
FCC /ABLE /		
FCC /RANGE/		
FCC / /		
FCC / /		
FCC / /		
FCB EOT		
BRA PNT311	*	GO TO PNT311
PT0305 PULS Y		
PNT30A INC ,S		
BRA PNT303		
PNT304 LEAS 1,S		
LDA ,S+	*	IF NOT PRIMFND
BNE PNT312		
	*	THEN
JSR CRLF	*	CRLF
JSR SCRSTR	*	WRITE('PRIMARY SENSOR NOT SPECIFIED . . . REENTER
FCC /PRIMA/	*	DATA ')
FCC /RY SE/		
FCC /NSOR /		
FCC /NOT S/		
FCC /PECIF/		

```

FCC /IED ./
FCC / . . /
FCC /REENT/
FCC /ER DA/
FCC /TA /
FCC / /
FCC / /
FCC / /
FCB EOT
BRA PNT300
* GO TO PNT300
* REPEAT
* CRLF
* WRITE('WHAT IS THE DESIRED STEADY-STATE ERROR?')

PNT312 JSR CRLF
JSR SCRSTR
FCC /WHAT /
FCC /IS TH/
FCC /E DES/
FCC /IRED /
FCC /STEAD/
FCC /Y-STA/
FCC /TE ER/
FCC /ROK?/
FCB EOT
JSR RDKEY
BVS PNT312
LDX #ERROR
JSR STORE
LEAU 4,U
JSR SETCUR
* RDKEY
* UNTIL V CLEAR
* REMOVE [U]
* SET OUTPUT CURRENT LIMIT
* REPEAT
* CRLF
* WRITE('WHAT IS THE LOAD RESISTANCE IN OHMS?')

PNT313 JSR CRLF
JSR SCRSTR
FCC /WHAT /
FCC /IS TH/
FCC /E LOA/
FCC /D RES/
FCC /ISTAN/
FCC /CE IN/
FCC / OHMS/
FCC /?/
FCB EOT
JSR RDKEY
BVS PNT313
LDX #LDRES
JSR STORE
JSR NLTPY
LEAX ,U
LDD #A800
PSHU D
LDD #A4861
PSHU D
LEAY ,U
JSR CMPXY
BLS PTO314
* RDKEY
* UNTIL V CLEAR
* LDRES := [U]
* [U] := PSCURR*LDRES
* IF [U] > 50000
* THEN

```

	LDX #PSMAX	*	PSMAX := 50000
	JSR STORE		
	LEAU 8,U	*	REMOVE [U],[U]
	BRA PNT315		
		*	ELSE
PTO314	LEAU 4,U	*	REMOVE [U]
	LDX #PSMAX	*	PSMAX := [U]
	JSR STORE		
	LEAU 4,U	*	REMOVE [U]
PNT315	CLR SWCHCR	*	INITIATE SWITCHES
	LDA #\$00	*	SET DDR
	STA SWITCH		
	LDA #\$0F	*	ENABLE SWITCH INTERRUPT
	STA SWCHCR		
	JSR RDSWCH	*	SET SWITCH STATUS
	ANDCC #\$00	*	ENABLE ALL INTERRUPTS
		*	INDICATE READY AND WAIT
		*	REPEAT
PNT314	JSR CRLF	*	CRLF
	JSR OUTSTR	*	WRITE('READY')
	FCC /READY/		
	FCB EOT		
	BRA PNT314	*	FOREVER

```

*****
*
*   THIS IS THE MAIN CONTROLLER PROGRAM.  IT IS CAPABLE OF CHANGING AND
*   CONTROLLING TEMPERATURE BASED UPON THE INPUT FROM THE THERMOMETERS, THE
*   OUTPUT TO THE HEATING COIL, AND THE INPUT FROM EITHER THE LSI-11 COMP-
*   UTER OR THE KEYPAD.  OTHER THAN THE CONTROL ITSELF, TWO OTHER FORMS OF
*   OUTPUT ARE PRODUCED:  (1) A SIGNAL TO THE LSI-11 INDICATING WHETHER THE
*   CONTROLLER IS CHANGING TEMPERATURE OR WHETHER IT IS MAINTAINING THE PRE-
*   SENT TEMPERATURE, AND (2) A DISPLAY OF THE NUMBER OF THE PRESENT THER-
*   MOMETER BEING MONITORED, THE LATEST DIRECTION OF CHANGE IN THE THERMOM-
*   ETER INPUT, AND THE PRESENT THERMOMETER INPUT.  THIS ROUTINE IS EXITED
*   ONLY FROM AN INTERRUPT OR A RESET.
*
*****

```

```

COMINT ORCC #$50      * MASK INTERRUPTS
LDS #SSTART          * RESET STACKS
LDU #USTART
LDA #$A0             * DISABLE A/D INPUT, HOLD A/D, DESELECT SENSORS
STA SENSEL
LDA #$3C             * SIGNAL INVALID
STA CONCR1
JSR RDCOMP           * READ COMPUTER INPUT
BRA CNTRL            * GO TO CNTRL

```

```

KEYINT ORCC #$50      * MASK INTERRUPTS
LDS #SSTART          * RESET STACKS
LDU #USTART
LDA #$A0             * DISABLE A/D INPUT, HOLD A/D, DESELECT SENSORS
STA SENSEL
LDA #$3C             * SIGNAL INVALID
STA CONCR1
JSR RDKEY            * READ KEYBOARD INPUT

```

```

CNTRL  ANDCC #$AF      * UNMASK INTERRUPTS
LDX #YD               * YD := {U}
JSR STORE
LEAU 4,U              * REMOVE {U}
LDA ROTARY            * IF ROTARY = 0
BNE PNT400

*   THEN
*   FOR I := 1 TO NUMSEN DO
PNT401 LDA ,S
CMP A NUMSEN
BHI PNT402
LDY #SEN1             *   Y := ADDR(SEN(I))
DEC A
LDB #$11
MUL
LEAY B,Y
LDA ,Y                *   IF SEN(I).TYPE = SECONDARY
ANDA #$40
BEQ PNT403

```

	*	THEN	
	*	IF YD > SEN(I).URNG1	
PSHS Y			
LDX #YD			
LEAY 1,Y			
JSR CMPXY			
BLS PNT404			
	*	THEN	
	*	IF YD <= SEN(I).URNG2	
LEAY 4,Y			
JSR CMPXY			
PULS Y			
BHI PNT403			
	*	THEN	
LEAS 1,S			
BRA PNT405		GO TO PNT405	
	*	ELSE	
PNT404 LEAY 4,Y	*	IF YD > SEN(I).URNG2	
JSR CMPXY			
PULS Y			
BLS PNT403			
	*	THEN	
LEAS 1,S			
BRA PNT405		GO TO PNT405	
PNT403 INC ,S			
BRA PNT401			
PNT402 LEAS 1,S			
	*	REPEAT	
PNT406 JSR CRLF	*	CRLF	
JSR SCRSTR	*	WRITE('SECONDARY SENSOR UNDETERMINED . . .	
FCC /SECON/	*	WHICH SENSOR SHOULD BE USED?')	
FCC /DARY /			
FCC /SENSO/			
FCC /R UND/			
FCC /ETERM/			
FCC /INED /			
FCC / . . ./			
FCC / WHIC/			
FCC /H SEN/			
FCC /SOR S/			
FCC /HOULD/			
FCC / BE U/			
FCC /SED? /			
FCB EOT			
ORCC #\$10	*	MASK INTERRUPT	
JSR RDDIG	*	RDDIG	
ANDCC #\$EF	*	UNMASK INTERRUPT	
BVS PNT406	*	UNTIL V CLEAR	
TST A	*	IF (A = 0) OR (A > NUMSEN)	
BEQ PNT407			
CMPA NUMSEN			
BLS PNT408			
	*	THEN	
PNT407 JSR CRLF	*	CRLF	
JSR SCRSTR	*	WRITE('INVALID INPUT	')
FCC /INVAL/			

```

FCC /ID IN/
FCC /PUT /
FCC / /
FCC / /
FCC / /
FCB EOT
BRA PNT406
PNT408 LDY #SEN1
DECA
LDB #S11
MUL
LEAY B,Y
BRA PNT405

PNT400 CNPA NUMSEN
BLS PNT409

LDY #SEN1
LDA NUMSEN
DECA
LDB #S11
MUL
LEAY B,Y
BRA PNT405

PNT409 LDY #SEN1
DECA
LDB #S11
MUL
LEAY B,Y
PNT405 LDA ,Y
ANDA #S07
DECA
LSLA
LSLA
LDB ,Y
ANDB #S40
BEQ PNT40A
ORA #S10
PNT40A ORA #S80
STA SECDRY
JSR PWRSEN
CLR ,-S
LEAS -12,S
PNT40B LDA 12,S
BEQ PNT40C

LDA SECDRY
STA SENSEL
BRA PNT40D

PNT40C LDA PRIMRY
STA SENSEL
PNT40D ANDA #S0C

```

```

* GO TO PNT406
* Y := ADDR(SEN(A))

* ELSE
* IF ROTARY > NUMSEN
* THEN
* Y := ADDR(SEN(NUMSEN))

* ELSE
* Y := ADDR(SEN(ROTARY))

* SECONDARY := SEN(Y).SIG:SEN(Y).NUM

* POWER SENSORS
* SECCNTRL := FALSE
* RESERVE N,K,SENNO, POL, YP,YPMI
* IF SECCNTRL
* THEN
* SENSEL := SECDRY

* ELSE
* SENSEL := PRIMRY
* Y := SENSOR(SENSEL)

```

LSRA	
LSRA	
LDB #S11	
MUL	
LDY #SEN1	
LEAY B,Y	
LDB ,Y	* POL := POL(SENSOR)
SEX	
STA 8,S	
ANDB #S07	* SENNO := SENNO(SENSOR)
STB 9,S	
LDA #SFF	* LET RELAYS BOUNCE
PNT40E DECA	
BNE PNT40E	
LDA SENSEL	* ENABLE A/D INPUT
ANDA #S7F	
STA SENSEL	
LDD SENDAO	* CLEAR A/D DATA FLAG
LDA SENSEL	* RUN A/D
ORA #S40	
STA SENSEL	
ORCC #S10	* MASK INTERRUPTS
LDD >U4+2	* U5 := U4
STD >U5+2	
LDD >U4	
STD >U5	
LDD >U3+2	* U4 := U3
STD >U4+2	
LDD >U3	
STD >U4	
LDD >U2+2	* U3 := U2
STD >U3+2	
LDD >U2	
STD >U3	
LDD >U1+2	* U2 := U1
STD >U2+2	
LDD >U1	
STD >U2	
LDD >U0+2	* U0 := U0
STD >U1+2	
LDD >U0	
STD >U1	
ANDCC #SEF	* UNMASK INTERRUPTS
JSR RDSN	* RDSN
LDA 12,S	* IF SECCNTRL
BEQ PNT40F	
	* THEN
LDX #YD	* YD := [U]
JSR STORE	
PNT40F LDX #YS	* YS := [U]
JSR STORE	
LEAX ,S	* YP := [U]
JSR STORE	
JSR CRLF	* CRLF

LDA 9,S	* WRITE(SENNUM)
JSR OUTCH	
JSR OUTS	* WRITE(' ')
JSR OUTS	
JSR COPY	* WRITE(YP)
JSR OUTDEC	
LDA 8,S	* IF POL = NEGATIVE
BPL PNT410	
	* THEN
LEAX ,U	* IF [U] = 0
JSR CMPXO	
BNE PNT411	
	* THEN
LEAU 4,U	* REMOVE [U]
JSR PMAXNO	* [U] := P.MAXNO
BRA PNT410	
	* ELSE
PNT411 JSR INVRSE	* [U] := 1/[U]
PNT410 LDX #Y0	* Y0 := [U]
JSR STORE	
LDX #Y1	* Y1 := [U]
JSR STORE	
LDX #Y2	* Y2 := [U]
JSR STORE	
LDX #Y3	* Y3 := [U]
JSR STORE	
LDX #Y4	* Y4 := [U]
JSR STORE	
LEAU 4,U	* REMOVE [U]
LDA 12,S	* IF SECCNTL
BEQ PNT412	
	* THEN
LDA #S13	* N := MAXN(19)
STA 11,S	
LDA #S34	* SIGNAL VALID
STA CONCR1	
BRA PNT413	
	* ELSE
PNT412 CLR 11,S	* N := 0
PNT413 CLR 10,S	* K := 0
	* REPEAT
PNT430 LDA SENSEL	* RUN A/D
ORA #S40	
STA SENSEL	
LDX #Y4	* IF (Y4 <> 0) AND (U5 <> 0)
JSR CMPXO	
BEQ PNT414	
LDX #U5	
JSR CMPXO	
BEQ PNT414	
	* THEN
LDX #U3	* [U] := (*5 POINT*)
JSR RECALL	* U3
LDD #S0000	

PSHU D	*	4
LDD #S41C0		
PSHU D		
JSR MLTPLY	*	*
LDX #U4	*	U4
JSR RECALL		
LDX #Y3	*	Y3
JSR RECALL		
JSR MLTPLY	*	*
LDX #Y4	*	Y4
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U4	*	U4
JSR RECALL		
JSR COPY	*	U4
JSR MLTPLY	*	*
LDX #U5	*	U5
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR NEGATE	*	CHS
JSR PLUS	*	+
LDX #U4	*	U4
JSR RECALL		
JSR COPY	*	U4
JSR MLTPLY	*	*
LDX #U5	*	U5
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR MLTPLY	*	*
LDX #U2	*	U2
JSR RECALL		
LDX #U4	*	U4
JSR RECALL		
JSR MLTPLY	*	*
LDX #U3	*	U3
JSR RECALL		
JSR COPY	*	U3
JSR MLTPLY	*	*
JSR PLUS	*	+
LDD #S0000	*	-3
PSHU D		
LDD #SC160		
PSHU D		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U3	*	U3
JSR RECALL		
LDX #Y3	*	Y3
JSR RECALL		
JSR MLTPLY	*	*

LDD #S0000	*	3
PSHU D		
LDD #S4160		
PSHU D		
JSR MLTPLY	*	*
LDX #U4	*	U4
JSR RECALL		
LDX #Y2	*	Y2
JSR RECALL		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U4	*	U4
JSR RECALL		
JSR MLTPLY	*	*
LDX #Y4	*	Y4
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR NEGATE	*	CHS
JSR PLUS	*	+
LDX #U4	*	U4
JSR RECALL		
JSR MLTPLY	*	*
LDX #U5	*	U5
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
LDX #U4	*	U4
JSR RECALL		
LDX #U1	*	U1
JSR RECALL		
JSR MLTPLY	*	*
LDX #U3	*	U3
JSR RECALL		
LDX #U2	*	U2
JSR RECALL		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDD #S0000	*	2
PSHU D		
LDD #S4140		
PSHU D		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U2	*	U2
JSR RECALL		
LDX #Y3	*	Y3
JSR MLTPLY	*	*
LDX #U3	*	U3
JSR RECALL		
LDX #Y2	*	Y2
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U4	*	U4

JSR RECALL		
JSR MLTPLY	*	*
LDD #S0000	*	2
PSHU D		
LDD #S4140		
PSHU D		
JSR MLTPLY	*	*
LDX #U3	*	U3
JSR RECALL		
JSR COPY	*	U3
JSR MLTPLY	*	*
LDX #Y3	*	Y3
JSR RECALL		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U4	*	U4
JSR RECALL		
JSR COPY	*	U4
JSR MLTPLY	*	*
LDX #Y1	*	Y1
JSR RECALL		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #Y4	*	Y4
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U5	*	U5
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
LDX #TABLE	*	Y+
LDA 11,S		
LDB #S04		
MUL		
LEAX B,X		
JSR RECALL		
LDX #YD		
JSR RECALL		
LDX #YS		
JSR RECALL		
JSR NEGATE		
JSR PLUS		
JSR MLTPLY		
LDX #YS		
JSR RECALL		
JSR PLUS		
LDA 8,S		
BPL PNT415		
LEAX ,U		
JSR CMPXO		
BNE PNT416		
LEAU 4,U		

	JSR PMAXNO		
	BRA PNT415		
PNT416	JSR INVRSE		
PNT415	LDX #U5	*	U5
	JSR RECALL		
	JSR MLTPLY	*	*
	LDX #U4	*	U4
	JSR RECALL		
	LDX #Y0	*	Y0
	JSR RECALL		
	JSR MLTPLY	*	*
	JSR NEGATE	*	CHS
	JSR PLUS	*	+
	LDX #U3	*	U3
	JSR RECALL		
	LDX #Y1	*	Y1
	JSR RECALL		
	JSR MLTPLY	*	*
	JSR NEGATE	*	CHS
	JSR PLUS	*	+
	LDX #U2	*	U2
	JSR RECALL		
	LDX #Y2	*	Y2
	JSR RECALL		
	JSR MLTPLY	*	*
	JSR NEGATE	*	CHS
	JSR PLUS	*	+
	LDX #U1	*	U1
	JSR RECALL		
	LDX #Y3	*	Y3
	JSR RECALL		
	JSR MLTPLY	*	*
	JSR NEGATE	*	CHS
	JSR PLUS	*	+
	LDX #Y4	*	Y4
	JSR RECALL		
	JSR INVRSE	*	1/X
	JSR MLTPLY	*	*
	JSR PLUS	*	+
	BRA PNT417		
		*	ELSE
PNT414	LDX #Y3	*	IF (Y3 <> 0) AND (U4 <> 0)
	JSR CMPXO		
	BEQ PNT418		
	LDX #U4		
	JSR CMPXO		
	BEQ PNT418		
		*	THEN
	LDX #U3	*	{U} := (*4 POINT*)
	JSR RECALL	*	U3
	JSR COPY	*	U3
	JSR MLTPLY	*	*
	LDX #U4	*	U4
	JSR RECALL		

JSR INVRSE	*	1/X
JSR MLTPLY	*	*
LDX #U3	*	U3
JSR RECALL		
LDX #Y2	*	Y2
JSR RECALL		
JSR MLTPLY	*	*
LDX #Y3	*	Y3
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR NEGATE	*	CHS
JSR PLUS	*	+
LDX #U2	*	U2
JSR RECALL		
LDD #S0000	*	-3
PSHU D		
LDD #S0160		
PSHU D		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U3	*	U3
JSR RECALL		
JSR COPY	*	U3
JSR MLTPLY	*	*
LDX #U4	*	U4
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR MLTPLY	*	*
LDX #U3	*	U3
JSR RECALL		
LDX #Y1	*	Y1
JSR RECALL		
JSR MLTPLY	*	*
LDX #U2	*	U2
JSR RECALL		
LDX #Y2	*	Y2
JSR RECALL		
JSR MLTPLY	*	*
LDD #S0000	*	2
PSHU D		
LDD #S4140		
PSHU D		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U3	*	U3
JSR RECALL		
JSR MLTPLY	*	*
LDX #Y3	*	Y3
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR PLUS	*	+

LDX #U2	*	U2
JSR RECALL		
JSR COPY	*	U2
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U1	*	U1
JSR RECALL		
LDX #U3	*	U3
JSR RECALL		
JSR MLTPLY	*	*
LDD #S0000	*	2
PSHU D		
LDD #S4140		
PSHU D		
JSR MLTPLY	*	*
JSR PLUS	*	+
LDX #U4	*	U4
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
LDX #TABLE	*	Y+
LDA 11,S		
LDB #S04		
MUL		
LEAX B,X		
JSR RECALL		
LDX #YD		
JSR RECALL		
LDX #YS		
JSR RECALL		
JSR NEGATE		
JSR PLUS		
JSR MLTPLY		
LDX #YS		
JSR RECALL		
JSR PLUS		
LDA 8,S		
BPL PNT419		
LEAX ,U		
JSR CMPX0		
BNE PNT41A		
LEAU 4,U		
JSR PHAX70		
BRA PNT419		
PNT41A JSR INVRSE		
PNT419 LDX #U4	*	U4
JSR RECALL		
JSR MLTPLY	*	*
LDX #U3	*	U3
JSR RECALL		
LDX #Y0	*	Y0
JSR RECALL		
JSR MLTPLY	*	*
JSR NEGATE	*	CHS

JSR PLUS	*	+
LDX #U2	*	U2
JSR RECALL		
LDX #Y1	*	Y1
JSR RECALL		
JSR MLTPLY	*	*
JSR NEGATE	*	CHS
JSR PLUS	*	+
LDX #U1	*	U1
JSR RECALL		
LDX #Y2	*	Y2
JSR RECALL		
JSR MLTPLY	*	*
JSR NEGATE	*	CHS
JSR PLUS	*	+
LDX #Y3	*	Y3
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR PLUS	*	+
BRA PNT417		
	*	ELSE
PNT418 LDX #Y2	*	IF (Y2 <> 0) AND (U3 <> 0)
JSR CMPX0		
BEQ PNT41B		
LDX #U3		
JSR CMPX0		
BEQ PNT41B		
	*	THEN
LDX #Y1	*	[U] := (*3 POINT*)
JSR RECALL	*	Y1
LDX #Y2	*	Y2
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
LDX #U2	*	U2
JSR RECALL		
LDX #U3	*	U3
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR NEGATE	*	CHS
JSR PLUS	*	+
LDX #U2	*	U2
JSR RECALL		
JSR MLTPLY	*	*
LDX #U1	*	U1
JSR RECALL		
LDD #S0000	*	2
PSHU D		
LDD #S4140		
PSHU D		
JSR MLTPLY	*	*
JSR PLUS	*	+

	LDX #U2	*	U2
	JSR RECALL		
	JSR MLTPLY	*	*
	LDX #U3	*	U3
	JSR RECALL		
	JSR INVRSE	*	1/X
	JSR MLTPLY	*	*
	LDX #TABLE	*	Y+
	LDA 11,S		
	LDB #S04		
	MUL		
	LEAX B,X		
	JSR RECALL		
	LDX #YD		
	JSR RECALL		
	LDX #YS		
	JSR RECALL		
	JSR NEGATE		
	JSR PLUS		
	JSR MLTPLY		
	LDX #YS		
	JSR RECALL		
	JSR PLUS		
	LDA 8,S		
	BPL PNT41C		
	LEAX ,U		
	JSR CMPXO		
	BNE PNT41D		
	LEAU 4,U		
	JSR PMAXNO		
	BRA PNT41C		
PNT41D	JSR INVRSE		
PNT41C	LDX #U3	*	U3
	JSR RECALL		
	JSR MLTPLY	*	*
	LDX #U2	*	U2
	JSR RECALL		
	LDX #Y0	*	Y0
	JSR RECALL		
	JSR MLTPLY	*	*
	JSR NEGATE	*	CHS
	JSR PLUS	*	+
	LDX #U1	*	U1
	JSR RECALL		
	LDX #Y1	*	Y1
	JSR RECALL		
	JSR MLTPLY	*	*
	JSR NEGATE	*	CHS
	JSR PLUS	*	+
	LDX #Y2	*	Y2
	JSR RECALL		
	JSR INVRSE	*	1/X
	JSR MLTPLY	*	*
	JSR PLUS	*	+

BRA PNT417	*	
	*	ELSE
PNT41B LDX #Y1	*	IF (Y1 <> 0) AND (U2 <> 0)
JSR CMPX0		
BEQ PNT41E		
LDX #U2		
JSR CMPX0		
BEQ PNT41E		
	*	THEN
LDX #U1	*	[U] := (*2 POINT*)
JSR RECALL	*	U1
JSR COPY	*	U1
JSR MLTPLY	*	*
LDX #U2	*	U2
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
LDX #TABLE	*	Y+
LDA 11,S		
LDB #S04		
MUL		
LEAX B,X		
JSR RECALL		
LDX #YD		
JSR RECALL		
LDX #YS		
JSR RECALL		
JSR NEGATE		
JSR PLUS		
JSR MLTPLY		
LDX #YS		
JSR RECALL		
JSR PLUS		
LDA 8,S		
BPL PNT41F		
LEAX ,U		
JSR CMPX0		
BNE PNT420		
LEAU 4,U		
JSR PLAXNO		
BRA PNT41F		
PNT420 JSR INVRSE		
PNT41F LDX #U2	*	U2
JSR RECALL		
JSR MLTPLY	*	*
LDX #U1	*	U1
JSR RECALL		
LDX #Y0	*	Y0
JSR RECALL		
JSR MLTPLY	*	*
JSR NEGATE	*	CHS
JSR PLUS	*	+
LDX #Y1	*	Y1
JSR RECALL		

JSR INVRSE	*	1/X
JSR MLTPLY	*	*
JSR PLUS	*	+
BRA PNT417		
	*	ELSE
PNT41E LDX #Y0	*	IF (Y0 <> 0) AND (U1 <> 0)
JSR CMPX0		
BEQ PNT421		
LDX #U1		
JSR CMPXU		
BEQ PNT421		
	*	THEN
LDX #TABLE	*	[U] := (*1 POINT*)
LDA 11,S	*	Y+
LDB #S04		
MUL		
LEAX B,X		
JSR RECALL		
LDX #YD		
JSR RECALL		
LDX #YS		
JSR RECALL		
JSR NEGATE		
JSR PLUS		
JSR MLTPLY		
LDX #YS		
JSR RECALL		
JSR PLUS		
LDA 8,S		
BPL PNT422		
LEAX ,U		
JSR CMPX0		
BNE PNT423		
LEAU 4,U		
JSR PMAXNO		
BRA PNT422		
PNT422 LDX #U1	*	U1
JSR RECALL		
JSR MLTPLY	*	*
LDX #Y0	*	Y0
JSR RECALL		
JSR INVRSE	*	1/X
JSR MLTPLY	*	*
BRA PNT417		
	*	ELSE
PNT421 LDD #S0000	*	[U] := 1
PSHU D		
LDD #S40C0		
PSHU D		
PNT417 JSR SETPWR	*	SETPWR
LDD >Y3+2	*	Y4 := Y3
STD >Y4+2		
LDD >Y3		
STD >Y4		

LDD >Y2+2	* Y3 := Y2
STD >Y3+2	
LDD >Y2	
STD >Y3	
LDD >Y1+2	* Y2 := Y1
STD >Y2+2	
LDD >Y1	
STD >Y2	
LDD >Y0+2	* Y1 := Y0
STD >Y1+2	
LDD >Y0	
STD >Y1	
LDD 2,S	* YPM1 := YP
STD 6,S	
LDD ,S	
STD 4,S	
ORCC #S10	* MASK INTERRUPTS
LDD >U4+2	* U5 := U4
STD >U5+2	
LDD >U4	
STD >U5	
LDD >U3+2	* U4 := U3
STD >U4+2	
LDD >U3	
STD >U4	
LDD >U2+2	* U3 := U2
STD >U3+2	
LDD >U2	
STD >U3	
LDD >U1+2	* U2 := U1
STD >U2+2	
LDD >U1	
STD >U2	
LDD >U0+2	* U1 := U0
STD >U1+2	
LDD >U0	
STD >U1	
ANDCC #SEF	* UNMASK INTERRUPTS
LDA 11,S	* IF N <> MAXN (19)
CMPIA #S13	
BEQ PNT424	
	* THEN
INC 11,S	* N := N + 1
PNT424 JSR RDSN	* RDSN
LDA SENSEL	* RUN A/D
ORA #S40	
STA SENSEL	
LEAX ,S	* YP := [U]
JSR STORE	
JSR CRLF	* CRLF
LDA 9,S	* WRITE(SENNUM)
JSR OUTCH	
LEAX ,S	* IF YP > YPM1
LEAY 4,S	

JSR CMPXY	
BLS PNT425	
LDA #'1	* THEN
JSR OUTCH	* WRITE('1')
BRA PNT426	
	* ELSE
PNT425 BEQ PNT427	* IF YP < YPM1
	* THEN
LDA #'1	* WRITE('1')
JSR OUTCH	
BRA PNT426	
	* ELSE
PNT427 LDA #'-	* WRITE('-')
JSR OUTCH	
PNT426 JSR OUTS	* WRITE(' ')
JSR COPY	* WRITE(YP)
JSR OUTDEC	
LDA 8,S	* IF POL = NEGATIVE
BPL PNT428	
	* THEN
LEAX ,U	* IF [U] = 0
JSR CMPX0	
BNE PNT429	
	* THEN
LEAU 4,U	* REMOVE [U]
JSR PMAXNO	* [U] := PMAXNO
BRA PNT428	
	* ELSE
PNT429 JSR INVRSE	* [U] := 1/[U]
PNT423 LDX #Y0	* Y0 := [U]
JSR STORE	
LEAU 4,U	* REMOVE [U]
LDA 12,S	* IF NOT SECCNTRL
BNE PNT430	
	* THEN
LEAX ,S	* IF ABS(YP - YD) <= ERROR
JSR RECALL	
LDX #YD	
JSR RECALL	
JSR NEGATE	
JSR PLUS	
LDA ,U	
ANDA #\$7F	
STA ,U	
LEAX ,U	
LDY #ERROR	
JSR CMPXY	
LEAU 4,U	
BHI PNT431	
	* THEN
INC 10,S	* K := K + 1
LDA 10,S	* IF K >= 5
CMPA #S05	

BLO PNT430	*	THEN
DEC 12,S	*	SECCTRL := TRUE
BRA PNT40B	*	
	*	ELSE
PNT431 CLR 10,S	*	K := 0
BRA PNT430	*	FOREVER

 *
 * 'TABLE' IS A TABLE OF THE MODEL THAT THE CONTROLLER IS TO FOLLOW WHEN *
 * CHANGING TEMPERATURE. THIS MODEL IS NORMALIZED (0 = STARTING POINT, *
 * 1 = ENDING POINT). THIS MODEL CONTAINS THE VALUES FOR A 20 POINT, CRI- *
 * TICALLY DAMPED CURVE. *
 *

TABLE FQB \$404848E6
 FQB \$4067BEA0
 FQB \$407571A6
 FQB \$407B67BC
 FQB \$407E0000
 FQB \$407F2124
 FQB \$407F9EFF
 FQB \$407FD5C7
 FQB \$407FED9F
 FQB \$407FF800
 FQB \$407FFC85
 FQB \$407FFE7C
 FQB \$407FFF57
 FQB \$407FFFB6
 FQB \$407FFFF0
 FQB \$407FFFF2
 FQB \$407FFFFA
 FQB \$407FFFFD
 FQB \$407FFFFF
 FQB \$40C00000

```

*****
*
*   THIS IS THE TERMINATING ROUTINE.  IT IS ENTERED ONLY THROUGH A 'I/O'
*   RESPONSE TO A DEFAULT PROMPT IN THE SWI ROUTINE.
*
*****

```

```

BYE   JSR CRLF      * CRLF
      JSR OUTSTR    * WRITE('BYE-BYE')
      FCC /BYE-B/
      FCC /YE/
      FCB EOT
PNT000 BRA PNT000

```


ORG SUB

```
*****
*
*   SUBROUTINE OUTCH OUTPUTS A CHARACTER TO THE DISPLAY LOCATION POINTED
*   TO BY THE DISPLAY POINTER(DISPAD). IT ALSO PLACES THE CHARACTER IN
*   THE DISPLAY TABLE AND INCREMENTS THE DISPLAY POINTER.
*
*   ENTRY:  ACCA CONTAINS THE CHARACTER
*
*   EXIT:   ACCA CONTAINS THE CHARACTER
*           DISPLAY TABLE UPDATED
*           DISPLAY POINTER INCREMENTED
*
*   VOLATILE REGISTERS:  B, X, CC
*
*   STACK USAGE:  S = 2 BYTES
*
*****
```

```
OUTCH  INC DISPAD
        LDX #DIG1
        LDB DISPAD
        ANDB #$0F
        STA B,X
        STA DISPDA
        RTS
```

```

*****
*
*   SUBROUTINE OUTSTR OUTPUTS A STRING OF CHARACTERS TO THE DISPLAY.
*
*   ENTRY:  STRING IS LOCATED IMMEDIATELY AFTER THE SUBROUTINE CALL AND
*           MUST BE TERMINATED WITH AN EOT CHARACTER ($FF).
*
*   EXIT:   EXECUTION RESUMES AT INSTRUCTION IMMEDIATELY FOLLOWING EOT
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S  = 4 BYTES
*
*****

```

```

OUTSTR LDA [,S]
      INC 1,S
      BNE PNT000
      INC ,S
PNT000 CMPA #$FF
      BEQ PNT001
      JSR OUTCH
      BRA OUTSTR
PNT001 RTS

```

```

*****
*
*   SUBROUTINE OUTHR OUTPUTS THE RIGHT HALF-BYTE OF THE HEX NUMBER CON-
*   TAINED IN ACCA TO THE DISPLAY.
*
*   ENTRY:  ACCA CONTAINS THE HEX NUMBER
*
*   EXIT:  ACCA CONTAINS THE HEX NUMBER
*
*   VOLATILE REGISTERS:  B, X, CC
*
*   STACK USAGE:  S = 5 BYTES
*
*****

```

```

OUTHR  PSHS A
        ANDA #SOF
        JSR OUTCH
        PULS A
        RTS

```

```

*****
*
*   SUBROUTINE OUTHL OUTPUTS THE LEFT HALF-BYTE OF THE HEX NUMBER CON-
*   TAINED IN ACCA TO THE DISPLAY.
*
*   ENTRY:  ACCA CONTAINS THE HEX NUMBER
*
*   EXIT:  ACCA CONTAINS THE HEX NUMBER
*
*   VOLATILE REGISTERS:  B, X, CC
*
*   STACK USAGE:  S = 5 BYTES
*
*****

```

```

OUTH  PSHS A
      LSRA
      LSRA
      LSRA
      LSRA
      JSR OUTH
      PULS A
      RTS

```

```

*****
*
*   SUBROUTINE OUTHEX OUTPUTS THE HEX NUMBER CONTAINED IN ACCA TO THE
*   DISPLAY.
*
*   ENTRY:  ACCA CONTAINS THE HEX NUMBER
*
*   EXIT:  ACCA CONTAINS THE HEX NUMBER
*
*   VOLATILE REGISTERS:  B, X, CC
*
*   STACK USAGE:  S = 7 BYTES
*
*****
OUTHEX JSR OUTHL
        JSR OUTHR
        RTS

```

```

*****
*
*   SUBROUTINE OUTS OUTPUTS A SPACE TO THE DISPLAY.
*
*   ENTRY:  NONE
*
*   EXIT:  ACCA CONTAINS A SPACE CHAR ($30)
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 4 BYTES
*
*****
OUTS  LDA #'
      JSR OUTCH
      RTS

```

```

*****
*
*   SUBROUTINE OUTNUM OUTPUTS THE REAL NUMBER LOCATED ON THE TOP OF THE
*   USER STACK TO THE DISPLAY IN THE FORMAT: +0.XXXXXX1+XX   THE NUMBER
*   ON THE STACK IS DESTROYED.
*
*   ENTRY:  REAL NUMBER IS CONTAINED IN THE TOP 4 BYTES OF THE USER STACK
*
*   EXIT:  NUMBER IS REMOVED FROM THE USER STACK
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 9 BYTES
*                  U = 2 BYTES
*
*****
OUTNUM LDD ,U          * IF [U] = 0
        BNE PNT002
                * THEN
                *   ADD SIGNS
        PSHU D
        BRA PNT003
                * ELSE
                *   EXPAND
PNT002 LDB ,U
        SEX
        PSHU A
        LSL 4,U
        ROL 3,U
        ROL 2,U
        ROL 1,U
        LDB 1,U
        SUBB #80
        SEX
        PSHU A
        BPL PNT004
        NEG B
PNT004 STB 2,U
PNT003 LDA 1,U          * WRITE(MANTISSA)
        BPL PNT005
        JSR OUTSTR
        FCC /+0./
        FCB EOT
        BRA PNT006
PNT005 JSR OUTSTR
        FCC /-0./
        FCB EOT
PNT006 LDA 3,U
        JSR OUTHEX
        LDA 4,U
        JSR OUTHEX
        LDA 5,U
        JSR OUTHEX
        LDA ,U          * WRITE(EXPONENT)
        BPL PNT007
        JSR OUTSTR

```

FCC /1+/
FCB EOT
BRA PT0008
PNT007 JSR OUTSTR
FCC /1-/
FCB EOT
PT0008 LDA 2,U
JSR OUTHEX
LEAU 6,U
RTS


```

*****
*
*   SUBROUTINE OUTDEC OUTPUTS THE REAL NUMBER LOCATED ON THE TOP OF THE
*   USER STACK TO THE DISPLAY IN THE DECIMAL FORMAT: +0.XXXXXX|+XX   THE
*   NUMBER ON THE STACK IS DESTROYED.
*
*   ENTRY:  REAL NUMBER IS CONTAINED IN THE TOP 4 BYTES OF THE USER STACK
*
*   EXIT:  NUMBER IS REMOVED FROM THE USER STACK
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 14 BYTES
*                  U = 25 BYTES
*
*****

```

```

OUTDEC JSR CONVRT
      LDA 1,U
      BMI PNT620
      JSR OUTSTR
      FCC /+0./
      FCB EOT
      BRA PNT621
PNT620 JSR OUTSTR
      FCC /-0./
      FCB EOT
PNT621 LDA 3,U
      JSR OUTHEX
      LDA 4,U
      JSR OUTHEX
      LDA 5,U
      JSR OUTHEX
      LDA ,U
      BMI PNT622
      JSR OUTSTR
      FCC /1+/
      FCCB EOT
      BRA PNT623
PNT622 JSR OUTSTR
      FCC /1-/
      FCB EOT
PNT623 LDA 2,U
      JSR OUTHEX
      LEAU 6,U
      RTS

```

```

*****
*
*   SUBROUTINE CRLF CLEARS THE DISPLAY.
*
*   ENTRY:  NONE
*
*   EXIT:  DISPLAY CONTAINS ALL BLANKS
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 5 BYTES
*
*****

```

```

CRLF  LDA #$10      * FOR I := 1 TO 16 DO
      PSHS A
      LDA #'        *  WRITE(' ')
PNT008 JSR OUTCH
      DEC ,S
      BNE PNT003
      LEAS 1,S
      LDA #$0F      * RESET(DISPAD)
      STA DISPAD
      RTS

```

```

*****
*
*   SUBROUTINE SCROCH SCROLLS ONE CHARACTER ONTO THE DISPLAY.  IT ALSO
*   UPDATES THE DISPLAY TABLE WITH THE CURRENT CONTENTS OF THE DISPLAY.
*
*   ENTRY:  ACCA CONTAINS THE CHARACTER
*
*   EXIT:  DISPLAY TABLE UPDATED
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 5 BYTES
*
*****

```

```

SCROCH PSHS A
      LDA #$8F          * RESET(DISPAD), DISABLE DISPLAY
      STA DISPAD
      LDX #DIG1+1      * FOR I := 2 TO 16 DO
PNT009 LDA ,X          *   SHIFT DIGITS
      PSHS X
      JSR OUTCH
      PULS X
      LEAX 1,X
      CMPX #DIG1+15
      BLS PNT009
      PULS A          * WRITE(A)
      JSR OUTCH
      LDA #$0F          * RESET(DISPAD), ENABLE DISPLAY
      STA DISPAD
      LDX #$27FF      * WAIT
PNT00A LEAX -1,X
      BNE PNT00A
      RTS

```

```

*****
*
*   SUBROUTINE SCRSTR SCROLLS A STRING OF CHARACTERS ACROSS THE DISPLAY.
*
*   ENTRY:  STRING IS LOCATED IMMEDIATELY AFTER THE SUBROUTINE CALL AND
*           MUST BE TERMINATED WITH AN EOT CHARACTER ($FF).
*
*   EXIT:   EXECUTION RESUMES AT INSTRUCTION IMMEDIATELY FOLLOWING EOT
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 9 BYTES
*
*****

```

```

SCRSTR LDA [,S]
      INC 1,S
      BNE PNT00B
      INC ,S
PNT00B CMPA #$FF
      BEQ PNT00C
      JSR SCROCH
      BRA SCRSTR
PNT00C RTS

```

```

*****
*
*   SUBROUTINE SCRIIR SCROLLS THE RIGHT HALF-BYTE OF THE HEX NUMBER CON-
*   TAINED IN ACCA ONTO THE DISPLAY.
*
*   ENTRY:  ACCA CONTAINS THE HEX NUMBER
*
*   EXIT:   ACCA CONTAINS THE HEX NUMBER
*
*   VOLATILE REGISTERS:  B, X, CC
*
*   STACK USAGE:  S = 10 BYTES
*
*****

```

```

SCRIIR  PSHS A
        ANDA #50F
        JSR SCROCH
        PULS A
        RTS

```

```

*****
*
*   SUBROUTINE SCRHL SCROLLS THE LEFT HALF-BYTE OF THE HEX NUMBER CON-
*   TAINED IN ACCA TO THE DISPLAY.
*
*   ENTRY:  ACCA CONTAINS THE HEX NUMBER
*
*   EXIT:   ACCA CONTAINS THE HEX NUMBER
*
*   VOLATILE REGISTERS:  B, X, CC
*
*   STACK USAGE:  S = 10 BYTES
*
*****

```

```

SCRHL  PSHS A
        LSRA
        LSRA
        LSRA
        LSRA
        JSR SCROCH
        PULS A
        RTS

```

```

*****
*
*   SUBROUTINE SCRHEX SCROLLS THE HEX NUMBER CONTAINED IN ACCA ONTO THE
*   DISPLAY.
*
*   ENTRY:  ACCA CONTAINS THE HEX NUMBER
*
*   EXIT:   ACCA CONTAINS THE HEX NUMBER
*
*   VOLATILE REGISTERS:  B, X, CC
*
*   STACK USAGE:  S = 12 BYTES
*
*****
SCRHEX JSR SCRHL
        JSR SCRHR
        RTS

```

```

*****
*
*   SUBROUTINE SCRS SCROLLS A SPACE ONTO THE DISPLAY.
*
*   ENTRY:  NONE
*
*   EXIT:   NONE
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 9 BYTES
*
*****
SCRS  LDA #'
      JSR SCROCH
      RTS

```



```

*****
*
*   SUBROUTINE SCRNUM SCROLLS THE REAL NUMBER LOCATED ON THE TOP OF THE
*   USER STACK ONTO THE DISPLAY IN THE FORMAT: +0.XXXXXX|+XX   THE NUMBER
*   ON THE STACK IS DESTROYED.
*
*   ENTRY:  REAL NUMBER IS CONTAINED IN THE TOP 4 BYTES OF THE USER STACK
*
*   EXIT:   NUMBER IS REMOVED FROM THE USER STACK
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 14 BYTES
*                  U = 2 BYTES
*
*****

```

```

*****
SCRNUM LDD ,U      * IF [U] = 0
      BNE PNTOOD

      * THEN
      PSHU D      *   ADD SIGNS
      BRA PNTOOE

      * ELSE
      PNTOOD LDB ,U *   EXPAND
      SEX
      PSHU A
      LSL 4,U
      ROL 3,U
      ROL 2,U
      ROL 1,U
      LDB 1,U
      SUBB #$80
      SEX
      PSHU A
      BPL PNTTOOF
      NEG B
PNTTOOF STB 2,U
PNTTOOE LDA 1,U   * WRITE(MANTISSA)
      BPL PNT010
      JSR SCRSTR
      FCC /+0./
      FCB EOT
      BRA PNT011
PNT010 JSR SCRSTR
      FCC /-0./
      FCB EOT
PNT011 LDA 3,U
      JSR SCRHEX
      LDA 4,U
      JSR SCRHEX
      LDA 5,U
      JSR SCRHEX
      LDA ,U      * WRITE(EXPONENT)
      BPL PNT012
      JSR SCRSTR

```

FCC /1+/
FCB EOT
BRA PT0013
PNT012 JSK SCRSTR
FCC /1-/
FCB EOT
PT0013 LDA 2,U
JSR SCRHEX
LEAU 6.U
RTS

```

*****
*
*   SUBROUTINE SCRDEC SCROLLS THE REAL NUMBER LOCATED ON THE TOP OF THE
*   USER STACK ONTO THE DISPLAY IN THE DECIMAL FORMAT: +0.XXXXXX|+XX
*   THE NUMBER ON THE STACK IS DESTROYED.
*
*   ENTRY:  REAL NUMBER IS CONTAINED IN THE TOP 4 BYTES OF THE USER STACK
*
*   EXIT:   NUMBER IS REMOVED FROM THE USER STACK
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 19 BYTES
*                  U = 25 BYTES
*
*****

```

```

*****
SCRDEC JSR CONVRT

```

```

        LDA 1,U
        BMI PNT630
        JSR SCRSTR
        FCC /+0./
        FCB EOT
        BRA PNT631
PNT630 JSR SCRSTR
        FCC /-0./
        FCB EOT
PNT631 LDA 3,U
        JSR SCRHEX
        LDA 4,U
        JSR SCRHEX
        LDA 5,U
        JSR SCRHEX
        LDA ,U
        BMI PNT632
        JSR SCRSCR
        FCC /|+ /
        FCB EOT
        BRA PNT633
PNT632 JSR SCRSTR
        FCC /|- /
        FCB EOT
PNT633 LDA 2,U
        JSR SCRHEX
        LEAU 6,U
        RTS

```

```

*****
*
*   SUBROUTINE RDSWCH READS THE CURRENT STATUS OF THE TOGGLE AND ROTARY
*   SWITCH OF THE TEMPERATURE CONTROLLER AND UPDATES THE SWITCH STATUS
*   LOCATIONS IN MEMORY. THIS SUBROUTINE ALSO SETS THE INTERRUPT CAPABILITY
*   OF THE KEYPAD AND THE COMPUTER TO CORRESPOND WITH THE TOGGLE SWITCH
*   STATUS.
*
*   ENTRY:  NONE
*
*   EXIT:   SWITCH STATUS MEMORY LOCATIONS UPDATED
*           SWITCH STATUS CONTAINED IN ACCD
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 15 BYTES
*
*****

```

```

RDSWCH LDA SWITCH      * READ(SWITCH)
      PSHS A
      ANDA #$60         * IF COMP/MAN = 'COMP'
      CMPA #$20
      BNE PNT013

      LDB #$00          * THEN
      BRA PNT014        *   ACCB := $00

      * ELSE
PNT013 CMPA #$40        *   IF COMP/MAN <> 'MAN'
      BEQ PNT015

      * THEN
      SWI              *   SWI(0)

PNT015 LDB #$FF        *   ACCB := $FF
PNT014 PULS A          * IF ROT = 'A'
      RORA
      BCS PNT016

      * THEN
      LDA #$00          *   ACCA := $00
      BRA PNT017

      * ELSE
PNT016 RORA            *   IF ROT = '1'
      BCS PNT018

      * THEN
      LDA #$01          *   ACCA := $01
      BRA PNT017

      * ELSE
PNT018 RORA            *   IF ROT = '2'
      BCS PNT019

      * THEN
      LDA #$02          *   ACCA := $02
      BRA PNT017

      * ELSE
PNT019 RORA            *   IF ROT = '3'
      BCS PNT01A

```

LDA #S03	*	THEN
BRA PNT017	*	ACCA := S03
	*	
PNT01A RORA	*	ELSE
BCS PNT01B	*	IF ROT = '4'
	*	
LDA #S04	*	THEN
BRA PNT017	*	ACCA := S04
	*	
PNT01B LDA #SFF	*	ELSE
PNT017 STD ROTARY	*	ACCA := SFF
	*	
PSHS D	*	ROTARY := [A]
TSTB	*	TOGGLE := [B]
BNE PTO01C	*	IF COMP/MAN = 'COMP'
	*	
LDA #S27	*	THEN
STA CONCRO	*	ENABLE COMP INT
LDA #S06	*	
STA KEYCR	*	DISABLE KEY INT
BRA PTO01D	*	
	*	
PT001C LDA #S04	*	ELSE
STA CONCRO	*	DISABLE COMP INT
LDA #S07	*	
STA KEYCR	*	ENABLE KEY INT
PT001D PULS D		
RTS		

```

*****
*
*   SUBROUTINE RDCOMP READS A NUMBER INPUT FROM THE COMPUTER AND PLACES
*   THE NUMBER ON THE USER STACK. THIS ROUTINE ALSO CHECK FOR UNDEFINED
*   COMPUTER INPUT AND IF SO GENERATES A SOFTWARE INTERRUPT.
*
*   ENTRY:  NONE
*
*   EXIT:  NUMBER PUSHED ONTO TOP 4 BYTES OF THE USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 4 BYTES
*                  U = 4 BYTES
*
*****
RDCOMP LDA COMCRO      * REPEAT
      BPL RDCOMP      *  READ(COMCRO)
                      * UNTIL DATA READY
      LDD COMDAO      *  READ(COMDAO)
      PSHS D
PNT01C LDA COMCRO      * REPEAT
      BPL PNT01C      *  READ(COMCRO)
                      * UNTIL DATA READY
      LDD COMDAO      * READ(COMDAO)
      PSHU D          * [U] := COMPUTER INPUT
      PULS D
      PSHU D
      COMD #$8000     * IF [U] = UNDEFINED
      BNE PNT01D
                      * THEN
                      *   SWI(1)
      SWI
      FCB $01
      LEAU 4,U        *  REMOVE [U]
      JMP ZERO        *   GO TO ZERO
                      * ELSE
PNT01D COMD #$0000     *   IF [U] <> 0
      BEQ PNT01E
                      * THEN
                      *   CORRECT FORMAT
      ASR 1,U
      ROR 2,U
      ROR 3,U
      LDA 1,U
      ORA #$40
      STA 1,U
PNT01E RTS

```

```

*****
*
*   SUBROUTINE PMAXNO PLACES THE LARGEST POSITIVE NUMBER WHICH THE REAL
*   NUMBER FORMAT CAN HANDLE ON THE USER STACK.
*
*   ENTRY:  NONE
*
*   EXIT:  TOP 4 BYTES OF USER STACK CONTAIN THE NUMBER
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*                  U = 4 BYTES
*
*****
PMAXNO LDD #$FFFF      * [U] := HEX +0.FFFFFE B+7F
      PSHU D
      LDA #$7F
      PSHU D
      RTS

```

```

*****
*
*   SUBROUTINE NMAXNO PLACES THE LARGEST NEGATIVE NUMBER WHICH THE REAL
*   NUMBER FORMAT CAN HANDLE ON THE USER STACK.
*
*   ENTRY:  NONE
*
*   EXIT:  TOP 4 BYTES OF USER STACK CONTAIN THE NUMBER
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*                  U = 4 BYTES
*
*****
NMAXNO LDD #FFFF      * [U] := HEX -0.FFFFFE B+7F
        PSHU D
        PSHU D
        RTS

```



```

*****
*
*   SUBROUTINE ZERO PLACES THE REAL NUMBER ZERO ON THE USER STACK
*
*   ENTRY:  NONE
*
*   EXIT:  TOP 4 BYTES OF USER STACK CONTAIN ZERO
*
*   VOLATILE RESIGTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*                  U = 4 BYTES
*
*****
ZERO  LDD #0000      * [U] := 0
      PSHU D
      PSHU D
      RTS

```

```

*****
*
*   SUBROUTINE LFTJST LEFT JUSTIFIES THE FRACTIONAL PART OF AN EXPANDED
*   REAL NUMBER AND SHRINKS THIS NUMBER BACK INTO THE STANDARD REAL NUMBER
*   FORMAT. THIS ROUTINE IS CALLED FROM ARITHMETIC ROUTINES.
*
*   ENTRY:   EXPANDED REAL NUMBER CONTAINED IN THE TOP 8 BYTES OF USER
*            STACK
*
*   EXIT:   REAL NUMBER CONTAINED IN TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 3 BYTES
*
*****
LFTJST LDD 2,U      * IF X-MAN <> 0
        BNE PNT01F
        LDD 4,U
        BNE PNT01F
        LDD 6,U
        BEQ PNT020

PNT01F CLR ,-S      *   THEN
        LDA 2,U      *   I := 0
        PNT022 BHI PNT021      *   WHILE X-MAN IS NOT LEFT JUSTIFIED DO
        INC ,S      *       I := I + 1
        ASL 7,U      *       X-MAN := X-MAN * 2
        ROL 6,U
        ROL 5,U
        ROL 4,U
        ROL 3,U
        ROL 2,U
        BRA PNT022

PNT021 LDA 1,U      *   X-EXP := X-EXP - I
        SUBA ,S+
        STA 1,U
        BHI PNT023

        LEAU 8,U
        JMP ZERO

PNT020 LEAU 8,U      *   IF X-EXP <= 0
        JMP ZERO      *   THEN
        *   REMOVE X
        *   GO TO ZERO

PNT023 LSR 2,U      *   ELSE
        ROR 3,U      *   REMOVE X
        ROR 4,U      *   GO TO ZERO
        BCC PNT024      *   ROUND X-MAN
        INC 4,U
        BNE PNT024
        INC 3,U
        BNE PNT024
        INC 2,U
        BPL PNT024      * IF N SET

```

INC 1,U	* THEN
BNE PNT025	* X-EXP := X-EXP + 1
	* IF X-EXP > 127
LDA ,U	* THEN
BMI PNT026	* IF X-SGN = POSITIVE
	* THEN
LEAU 8,U	* REMOVE X
JMP PMAXNO	* GO TO PMAXNO
	* ELSE
PNT026 LEAU 8,U	* REMOVE X
JMP NMAXNO	* GO TO NMAXNO
	* ELSE
PNT025 LSR 2,U	* X-MAN := X-MAN / 2
ROR 3,U	
ROR 4,U	
PNT024 LDD 3,U	* CONDENSE X
STD 6,U	
LDD 1,U	
LSLB	
ROR ,U	
RORA	
RORB	
STD 4,U	
LEAU 4,U	
RTS	

```

*****
*
*   SUBROUTINE SHRINK SHRINKS AN EXPANDED REAL NUMBER BACK INTO THE
*   STANDARD NUMBER FORMAT.
*
*   ENTRY:  EXPANDED NUMBER CONTAINED IN TOP 8 BYTES OF USER STACK
*
*   EXIT:   REAL NUMBER CONTAINED IN TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*
*****

```

```

SHRINK LDD 3,U      * CONDENSE X
        STD 6,U
        LDD 1,U
        LSR ,U
        RORA
        RORB
        STD 4,U
        ROR 6,U
        ROR 7,U
        LEAU 4,U
        RTS

```

```

*****
*
*   SUBROUTINE RECALL PUSHES THE REAL NUMBER POINTED TO BY THE X INDEX
*   POINTER ONTO THE USER STACK.
*
*   ENTRY:  X POINTER POINTS TO FIRST LOCATION OF REAL NUMBER
*
*   EXIT:  REAL NUMBER PUSHED ONTO USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*                  U = 4 BYTES
*
*****

```

```

RECALL LDD 2,X
        PSHU D
        LDD ,X
        PSHU D
        RTS

```

```

*****
*
*   SUBROUTINE STORE STORES THE REAL NUMBER LOCATED ON THE TOP OF THE
*   STACK AT THE FOUR BYTES POINTED TO BY THE X INDEX POINTER.  THE NUMBER
*   ON THE USER STACK REMAINS UNCHANGED.
*
*   ENTRY:  NUMBER CONTAINED IN TOP 4 BYTES OF USER STACK
*           X POINTER CONTAINS STORAGE ADDRESS
*
*   EXIT:   NUMBER CONTAINED IN TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*
*****
STORE  LDD 2,U
        STD 2,X
        LDD ,U
        STD ,X
        RTS

```

```

*****
*
*   SUBROUTINE COPY PUSHES AN ADDITIONAL COPY OF THE REAL NUMBER LOCATED
*   ON THE TOP OF THE USER STACK ONTO THE USER STACK.
*
*   ENTRY:  NUMBER CONTAINED IN TOP 4 BYTES OF USER STACK
*
*   EXIT:   TWO COPIES OF NUMBER CONTAINED IN TOP 8 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*                  U = 4 BYTES
*
*****
COPY  LDD 2,U
      PSHU D
      LDD 2,U
      PSHU D
      RTS

```

```

*****
*
*   SUBROUTINE XINTGY EXCHANGES THE TWO REAL NUMBERS LOCATED ON THE TOP
*   OF THE USER STACK.
*
*   ENTRY:  TWO NUMBERS CONTAINED IN TOP 8 BYTES OF USER STACK
*
*   EXIT:   TWO NUMBERS EXCHANGED
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 2 BYTES
*
*****

```

```

XINTGY LDD ,U
        LDX 4,U
        STD 4,U
        STX ,U
        LDD 2,U
        LDX 6,U
        STD 6,U
        STX 2,U
        RTS

```



```

*****
*
*   SUBROUTINE NEGATE CHANGES THE SIGN OF THE REAL NUMBER LOCATED ON THE
*   TOP OF THE USER STACK.
*
*   ENTRY:  REAL NUMBER CONTAINED IN THE TOP 4 BYTES OF USER STACK
*
*   EXIT:   SIGN CHANGED REAL NUMBER CONTAINED IN THE TOP 4 BYTES OF USER
*           STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 2 BYTES
*
*****
NEGATE LDD ,U      * IF X <> 0
          BEQ PNT03D
          * THEN
          EORA #$80 *   X := -X
          STA ,U
PNT03D RTS

```

```

*****
*
*   SUBROUTINE INVRSE REPLACES THE REAL NUMBER ON THE TOP OF THE USER
*   STACK WITH ITS MULTIPLICATIVE INVERSE.
*
*   ENTRY:  NUMBER CONTAINED IN TOP 4 BYTES OF USER STACK
*
*   EXIT:   INVERSE OF NUMBER CONTAINED IN TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 7 BYTES
*                  U = 12 BYTES
*
*****

```

```

INVRSE LDD ,U      * IF X = 0
      BNE PNT027

      * THEN
      *   SWI(2)

      SWI
      FCB $02
      LEAU 4,U     * REMOVE X
      JMP PMAXNO   * GO TO PMAXNO
PNT027 LDB ,U      * EXPAND X
      SEX
      PSIH A
      LEAU -6,U
      LDD 9,U
      ASL B
      ROLA
      STD 1,U
      LDD 7,U
      ROLB
      ROLA
      STB ,U
      STA 7,U
      LDD #$0000
      STD 3,U
      STA 5,U
      NEG 7,U      * R-EXP := - X-EXP
      LDD #$0000   * N-MAN := $800000000000
      PSHU D
      PSHU D
      LDA #$80
      PSHU D
      LDA #$19     * FOR I := 1 TO 25 DO
      PSHS A
PNT02A LDD 4,U      * N-MAN := N-MAN - X-MAN
      SUBD 10,U
      STD 4,U
      LDD 2,U
      SBCB 9,U
      SECA 8,U
      STD 2,U
      LDD ,U

```

SBCB 7,U	
SBCA 6,U	
STD ,U	
BCC PNT028	* IF N-MAN < 0
	* THEN
LDD 4,U	* N-MAN := N-MAN + X-MAN
ADDD 10,U	
STD 4,U	
LDD 2,U	
ADCB 9,U	
ADCA 8,U	
STD 2,U	
LDD ,U	
ADCB 7,U	
ADCA 6,U	
STD ,U	
ASL 16,U	* ASL R-MAN
ROL 15,U	
ROL 14,U	
BRA PNT029	
	* ELSE
PNT028 ORCC #S01	* SET CARRY
ROL 16,U	* ROL R-MAN
ROL 15,U	
ROL 14,U	
PNT029 LSR 6,U	* LSR X-MAN
ROR 7,U	
ROR 8,U	
ROR 9,U	
ROR 10,U	
ROR 11,U	
DEC ,S	
BNE PNT02A	
LEAS 1,S	
LEAU 12,U	* REMOVE N-MAN, X-MAN
LSR 2,U	* ROUND R-MAN
ROR 3,U	
ROR 4,U	
BCC PNT02B	
INC 4,U	
BNE PNT02B	
INC 3,U	
BNE PNT02B	
INC 2,U	
PNT02B LDD 2,U	* IF (N CLEAR) AND (R-MAN <> 0)
BMI PNT02C	
BNE PNT02D	
LDA 4,U	
BEQ PNT02C	
	* THEN
PNT02D INC 1,U	* R-EXP := R-EXP + 1
BNE PNT02E	* IF R-EXP > 127
	* THEN
LDA ,U	* IF R-SGN = POSITIVE

BMI PNT02F	*	THEN
LEAU 5,U	*	REMOVE R
JMP PMAXNO	*	GO TO PMAXNO
	*	ELSE
PNT02F LEAU 5,U	*	REMOVE R
JMP NMAXNO	*	GO TO NMAXNO
	*	ELSE
PNT02C LDA 1,U	*	R-EXP := R-EXP + 2
ADDA #\$02		
STA 1,U		
BCC PNT030	*	IF R-EXP > 127
	*	THEN
LDA ,U	*	IF R-SGN = POSITIVE
BMI PNT031		
	*	THEN
LEAU 5,U	*	REMOVE R
JMP PMAXNO	*	GO TO PMAXNO
	*	ELSE
PNT031 LEAU 5,U	*	REMOVE R
JMP NMAXNO	*	GO TO NMAXNO
	*	ELSE
PNT030 LDA #\$40	*	R-MAN := \$400000
STA 2,U		
PNT02E LDD 1,U	*	CONDENSE R
LSLB		
LSR ,U+		
RORA		
RORB		
STD ,U		
RTS		

```

*****
*
*   SUBROUTINE PLUS REPLACE THE TWO REAL NUMBERS LOCATED ON THE TOP OF
*   THE USER STACK WITH THEIR ARITHMETIC SUM.
*
*   ENTRY:  TWO NUMBERS CONTAINED IN THE TOP 8 BYTES OF USER STACK
*
*   EXIT:  SUM CONTAINED IN THE TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 5 BYTES
*                  U = 8 BYTES
*
*****
PLUS  LEAU -8,U      * EXPAND X
      LDB 8,U
      SEX
      STA ,U
      LSL 11,U
      ROL 10,U
      ROL 9,U
      ROL 8,U
      LDD 8,U
      STD 1,U
      LDD 10,U
      STD 3,U
      LDB 12,U      *EXPAND Y
      SEX
      STA 8,U
      LSL 15,U
      ROL 15,U
      ROL 13,U
      ROL 12,U
      LDD 12,U
      STD 9,U
      LDD 14,U
      STD 11,U
      LDD #0000
      STD 5,U
      STA 7,U
      STD 13,U
      STA 15,U
      LDD 1,U      * CASE ABS(X) - ABS(Y) OF
      CMPD 9,U
      BHI PNT032
      BLO PNT033
      LDD 3,U
      CMPD 11,U
      BHI PNT032
      BLO PNT033
      LDA ,U      * ABS(X) - ABS(Y): IF X-SGN EXOR Y-SGN = 1
      EORA 8,U
      BEQ PNT034

```

LEAU 16,U	*	THEN
JMP ZERO	*	REMOVE X, Y
	*	GO TO ZERO
PNT034 LEAU 8,U	*	ELSE
INC 1,U	*	REMOVE X
BEQ PNT035	*	Y-EXP := Y-EXP + 1
	*	IF Y-EXP <= 127.
	*	THEN
JMP SHRINK	*	GO TO SHRINK
	*	ELSE
PNT035 LDA ,U	*	IF Y-SGN = POSITIVE
BMI PNT036	*	
	*	THEN
LEAU 8,U	*	REMOVE Y
JMP PMAXNO	*	GO TO PMAXNO
	*	ELSE
PNT036 LEAU 3,U	*	REMOVE Y
JMP NMAXNO	*	GO TO NMAXNO
PNT032 LDD ,U	*	ABS(X) > ABS(Y): EXCHANGE X AND Y
LDX 8,U		
STD 8,U		
STX ,U		
LDD 2,U		
LDX 10,U		
STD 10,U		
STX 2,U		
LDD 4,U		
LDX 12,U		
STD 12,U		
STX 4,U		
PNT033 LDA 9,U	*	GO TO ABS(X) < ABS(Y)
SUBA 1,U	*	ABS(X) < ABS(Y): IF Y-EXP - X-EXP > 24
CMPA #S18		
BLS PNT037		
	*	THEN
LEAU 8,U	*	REMOVE X
JMP SHRINK	*	GO TO SHRINK
	*	ELSE
PNT037 TSTA	*	IF Y-EXP - X-EXP <> 0
BEQ PNT038		
	*	THEN
PSHS A	*	FOR I := 1 TO Y-EXP - X-EXP
PNT039 LSR 2,U	*	X-MAN := X-MAN / 2
ROR 3,U		
ROR 4,U		
ROR 5,U		
ROR 6,U		
ROR 7,U		
DEC ,S		
BNE PNT039		
LEAS 1,S		
PNT038 LDA ,U	*	IF Y-SGN EXOR X-SGN = 0
EORA 8,U		

BNE PNT03A	*	THEN
LDD 14,U	*	Y-MAN := Y-MAN + X-MAN
ADDD 6,U		
STD 14,U		
LDD 12,U		
ADCB 5,U		
ADCA 4,U		
STD 12,U		
LDD 10,U		
ADCB 3,U		
ADCA 2,U		
STD 10,U		
BCC PNT03B	*	IF C SET
	*	THEN
ROR 10,U	*	ROR Y-MAN
ROR 11,U		
ROR 12,U		
ROR 13,U		
ROR 14,U		
ROR 15,U		
INC 9,U	*	Y-EXP := Y-EXP + 1
BNE PNT03B	*	IF Y-EXP > 127
	*	THEN
LDA 8,U	*	IF Y-SGN = POSITIVE
BMI PNT03C		
	*	THEN
LEAU 16,U	*	REMOVE X, Y
JMP PMAXNO	*	GO TO PMAXNO
	*	ELSE
PNT03C LEAU 16,U	*	REMOVE X, Y
JMP NMAXNO	*	GO TO NMAXNO
PNT03B LEAU 8,U	*	REMOVE X
JMP SHRINK	*	GO TO SHRINK
	*	ELSE
PNT03A LDD 14,U	*	Y-MAN := Y-MAN - X-MAN
SUBD 6,U		
STD 14,U		
LDD 12,U		
SBCB 5,U		
SBCA 4,U		
STD 12,U		
LDD 10,U		
SBCB 3,U		
SBCA 2,U		
STD 10,U		
LEAU 8,U	*	REMOVE X
JMP LFTJST	*	GO TO LFTJST

```

*****
*
*   SUBROUTINE MLTPLY REPLACES THE TWO REAL NUMBERS LOCATED ON THE TOP OF
*   THE USER STACK WITH THEIR ARITHMETIC PRODUCT.
*
*   ENTRY:  TWO NUMBERS CONTAINED IN THE TOP 8 BYTES OF USER STACK
*
*   EXIT:   PRODUCT CONTAINED IN THE TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 5 BYTES
*                  U = 10 BYTES
*
*****

```

```

MLTPLY LDD ,U          * IF (X=0) OR (Y=0)
      BEQ PNT03E
      LDD 4,U
      BNE PNT03F

      * THEN
      *   REMOVE X, Y
      *   GO TO ZERO
      * ELSE
      *   ADD MEMORY
      *   EXPAND X

PNT03E LEAU 8,U
      JMP ZERO

PNT03F LEAU -10,U
      LDB 10,U
      SEX
      STA ,U
      LDD 12,U
      LSLB
      ROLA
      STD 3,U
      LDD 10,U
      ROLB
      ROLA
      STD 1,U
      LDB 14,U      *   EXPAND Y
      SEX
      STA 5,U
      LDD 16,U
      LSLB
      ROLA
      STD 8,U
      LDD 14,U
      ROLB
      ROLA
      STD 6,U
      LDA ,U      *   R-SGN := X-SGN EXOR Y-SGN
      EORA 5,U
      STA 10,U
      LDA #500    *   R-EXP := X-EXP + Y-EXP - 128
      LDB 1,U
      ADDB 6,U
      ROLA
      SUBD #50080

```


BLS PNT040	*	IF R-EXP > 0
TSTA	*	THEN
BEQ PNT041	*	IF R-EXP > 127
LDA 10,U	*	THEN
BMI PNT042	*	IF R-SGN = POSITIVE
LEAU 18,U	*	THEN
JMP PMAXNO	*	REMOVE X, Y, R
	*	GO TO PMAXNO
	*	ELSE
PNT042 LEAU 18,U	*	REMOVE X, Y, R
JMP NMAXNO	*	GO TO NMAXNO
	*	ELSE
PNT040 LEAU 18,U	*	REMOVE X, Y, R
JMP ZERO	*	GO TO ZERO
PNT041 STB 11,U	*	
LDD #0000	*	R-MAN := X-MAN * Y-MAN
STD 12,U		
STD 14,U		
LDA 4,U		
LDB 9,U		
MUL		
STD 16,U		
LDA 3,U		
LDB 9,U		
MUL		
ADDD 15,U		
STD 15,U		
LDA 4,U		
LDB 8,U		
MUL		
ADDD 15,U		
STD 15,U		
BCC PNT043		
INC 14,U		
PNT043 LDA 2,U		
LDB 9,U		
MUL		
ADDD 14,U		
STD 14,U		
LDA 3,U		
LDB 8,U		
MUL		
ADDD 14,U		
STD 14,U		
BCC PNT044		
INC 13,U		
PNT044 LDA 4,U		
LDB 7,U		
MUL		
ADDD 14,U		
STD 14,U		
BCC PNT045		

	INC 13,U	
PNT045	LDA 2,U	
	LDB 8,U	
	MUL	
	ADD 13,U	
	STD 13,U	
	BCC PNT046	
	INC 12,U	
PNT046	LDA 3,U	
	LDB 7,U	
	MUL	
	ADDD 13,U	
	STD 13,U	
	BCC PNT047	
	INC 12,U	
PNT047	LDA 2,U	
	LDB 7,U	
	MUL	
	ADDD 12,U	
	STD 12,U	
	LEAU 10,U	* REMOVE X, Y
	JMP LFTJST	* GO TO LFTJSR

```

*****
*
*   SUBROUTINE CMPXY COMPARES THE REAL NUMBER POINTED TO BY THE X INDEX
*   POINTER WITH THE REAL NUMBER POINTED TO BY THE Y INDEX POINTER AND SETS
*   THE C AND Z BITS OF THE CONDITION CODE REGISTER SO THAT CONDITIONAL
*   BRANCHES MAY BE USED.
*
*   ENTRY:  X POINTER POINTS TO FIRST REAL NUMBER
*           Y POINTER POINTS TO SECOND REAL NUMBER
*
*   EXIT:   C AND Z BITS OF CC REGISTER SET
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 4 BYTES
*
*****

```

```

CMPXY  LDD ,X          * IF X >= 0
      BMI PNT04B

      * THEN

      PSHS D
      LDD ,Y          *   IF Y < 0
      *               *   THEN
      BMI PNT049      *   GO TO PNT049
      *               *   ELSE
      CMPD ,S++       *   CASE ABS(X)___ABS(Y) OF
      BLO PNT04A      *   ABS(X) > ABS(Y): GO TO PNT04A
      BHI PNT04B      *   ABS(X) < ABS(Y): GO TO PNT04B
      *               *   ABS(X) = ABS(Y): GO TO PNT04C

      LDD 2,Y
      CMPD 2,X
      BLO PNT04A
      BHI PNT04B
      BRA PNT04C

      * ELSE

PNT048 ANDA #$7F
      PSHS D
      LDD ,Y          *   IF Y >= 0
      *               *   THEN
      BPL PNT04D      *   GO TO PNT04D
      *               *   ELSE
      CMPD ,S++       *   CASE ABS(X)___ABS(Y) OF
      BLO PNT04B      *   ABS(X) > ABS(Y): GO TO PNT04B
      BHI PNT04A      *   ABS(X) < ABS(Y): GO TO PNT04A
      *               *   ABS(X) = ABS(Y): GO TO PNT04C

      LDD 2,Y
      CMPD 2,X
      BLO PNT04B
      BHI PNT04A

PNT04C ANDCC #$FE      * (* X = Y *) C := 0
      ORCC #$04        *               Z := 1
      RTS

PNT049 LEAS 2,S
PNT04A ANDCC #$FA      * (* X > Y *) Z := 0

```

RTS
PNT04D LEAS 2,S
PNT04B ANDCC #SFB
ORCC #S01
RTS

* C := 0
* (* X < Y *) Z := 0
* C := 1

```

*****
*
*   SUBROUTINE CMPX0 COMPARES THE REAL NUMBER POINTED TO BY THE X INDEX
*   POINTER WITH ZERO AND SETS THE C AND Z BITES OF THE CONDITION CODE
*   REGISTER SO THAT CONDITIONAL BRANCHES MAY BE USED.
*
*   ENTRY:  X POINTER POINTS TO THE REAL NUMBER
*
*   EXIT:   C AND Z BITS OF CC REGISTER SET
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 2 BYTES
*
*****
CMPX0  LDD ,X          * IF X >= 0
      BHI PNT04E
      * THEN
      BNE PNT04F      * IF X = 0
      * THEN
      ANDCC #$FE      * C := 0
      ORCC #$04       * Z := 1
      RTS
      * ELSE
PNT04F ANDCC #$FA      * C := 0
      * Z := 0
      RTS
      * ELSE
PNT04E ORCC #$01      * C := 1
      ANDCC #$FB      * Z := 0
      RTS

```

```

*****
*
*   SUBROUTINE CONVRT CONVERTS THE REAL NUMBER ON THE TOP OF THE USER
*   STACK INTO A DECIMAL FORMAT SUITABLE FOR OUTPUTING TO THE DISPLAY.
*
*   ENTRY:  REAL NUMBER CONTAINED IN TOP 4 BYTES OF THE USER STACK
*
*   EXIT:   NUMBER IN DECIMAL FORMAT CONTAINED IN THE TOP 6 BYTES OF THE
*           USER STACK IN THE FOLLOWING ORDER:  SIGN OF THE EXPONENT, SIGN OF
*           THE MANTISSA, THE EXPONENT, AND THE MANTISSA
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 12 BYTES
*                 U = 25 BYTES
*
*****

```

```

*****
CONVRT LDD ,U      * IF [U] = 0
          BNE PNT600
          * THEN
          PSHU D      * ADD SIGNS
          RTS
          * ELSE
PNT600 LDB ,U      * EXPAND [U]
          SEX
          PSHU A
          LSL 4,U
          ROL 3,U
          ROL 2,U
          ROL 1,U
          LDB 1,U
          SUBB #$80
          SEX
          PSHU A
          BPL PNT601
          NEGB
PNT601 STB 2,U
          BNE PFX000
          * IF B = 0
          * THEN
          JSR ZERO
          BRA PFB5A
          * ELSE
          * FORM B ON U
PFX000 LDD #$0000
          PSHU D
          LDB 4,U
          LDA #$38
PNT602 DECA
          LSLB
          BPL PNT602
          ASR 2,U
          RORA
          RORB
          PSHU D
PFB5A LDD #$104D      * LOG(2)

```

PSHU D		
LDD #3FCD		
PSHU D		
JSR MLTPLY	*	*
JSR COPY	*	COPY
LDB ,U	*	EXPAND
SEX		
PSHU A		
LSL 4,U		
ROL 3,U		
ROL 2,U		
ROL 1,U		
LDA 1,U	*	IF ([U]EXP - \$80) < 0
SUBA #80		
BHS PNT603		
	*	THEN
LEAU 5,U	*	REMOVE [U]
JSR ZERO	*	[U] := 0
CLR 10,U	*	D := 0
BRA PNT604		
	*	ELSE
PNT603 BNE PNT605	*	IF([U]EXP - \$80) = 0
	*	THEN
LDD #0000	*	[U]EXPMAN := 1
STD 3,U		
LDD #8180		
STD 1,U		
LDA #01	*	D := 1
STA 11,U		
BRA PNT606		
	*	ELSE
PNT605 CLR ,-U	*	ZERO RESULT
PSHS A	*	FOR I := ([U]EXP - \$80) DOWNT0 1 DO
PNT607 LSL 3,U	*	LSL [U]MAN
ROL ,U	*	ROL RESULT
DEC ,S		
BNE PNT607		
LEAS 1,S		
LDA ,U	*	ROUND RESULT
LSL 3,U		
ADCA #00		
STA ,U		
STA 12,U	*	D := RESULT
LDD #0000	*	ZERO [U]MAN
STD 3,U		
STA 5,U		
LDB ,U	*	FORM D ON U
LDA #88		
PFX002 DECA		
LSLB		
BPL PFX002		
STD 2,U		
LEAU 1,U		
PNT606 LSR ,U+	*	SHRINK

	ROR ,U		
	ROR 1,U		
	ROR 2,U		
	ROR 3,U		
PNT603	LDA 10,U	*	CONVERT D TO DEC
	PSHU A		
	CLR 11,U		
	LDA #S08		
	PSHS A		
PNT609	LSL ,U		
	LDA 11,U		
	ADCA 11,U		
	DAA		
	STA 11,U		
	DEC ,S		
	BNE PNT609		
	LEAS 1,S		
	LEAU 1,U		
	JSR NEGATE	*	NEGATE
PNT604	JSR PLUS	*	+
	LDD #SAEC7	*	LN(10)
	PSHU D		
	LDD #S4149		
	PSHU D		
	JSR MLTPLY	*	*
	LEAS -4,S	*	(* EXP([U]) *)
	LEAX ,S		
	JSR STORE		
	LEAU 4,U	*	REMOVE [U]
	LDD #S0000	*	1
	PSHU D		
	LDD #S40C0		
	PSHU D		
	LDA #SOC	*	FOR I := 12 DOWNT0 1 DO
	PSHS A		
PNT60C	LEAX 1,S	*	X
	JSR RECALL		
	LDD #S0000	*	I
	PSHU D		
	LDA #S88		
	LDB ,S		
PNT60D	DECA		
	LSLB		
	BPL PNT60D		
	LSRA		
	RORB		
	PSHU D		
	JSR INVRSE	*	1/X
	JSR MLTPLY	*	*
	JSR MLTPLY	*	*
	LDD #S0000	*	1
	PSHU D		
	LDD #S40C0		
	PSHU D		


```

JSR PLUS      *      +
DEC ,S
BNE PNT60C
LEAS 5,S
LDD 8,U      *      FORM A ON U
PSHU D
LDB 9,U
LDA #$80
LSRA
RORB
PSHU D
ROR 2,U
ROR 3,U
JSR MLTPLY   *      *
LDD ,U      *      IF [U] >= 1
LSLB
ROLA
CMPA #$81
BLO PNT60E
                *      THEN
LDD #$6666   *      [U] := [U] / 10
PSHU D
LDD #$3EE6
PSHU D
JSR MLTPLY
LDA 4,U      *      IF D-SGN < 0
BMI PFX001
                *      THEN
LDA 6,U      *      ABS(D) := ABS(D) + 1
ADDA #S01
BRA PFB6
                *      ELSE
LDA 6,U      *      ABS(D) := ABS(D) - 1
ADDA #$99
PFB6 DAA
STA 6,U
                *      (* CONVERT C *)
                *      EXPAND
PNT60E LSL 3,U
ROL 2,U
ROL 1,U
ROL ,U
LDA #$80      *      IF ($80 - [U]EXP) <> 0
SUBA ,U+
BEQ PNT60F
                *      THEN
                *      FOR I := ($80 - [U]EXP) DOWNT0 1 DO
PNT610 LSR ,U      *      LSR [U]MAN
ROR 1,U
ROR 2,U
DEC ,S
BNE PNT610
LEAS 1,S
PNT60F LDD #$0000 *      ZERO RESULT
STD 6,U

```

	STA 8,U		
	LDA #S18	*	FOR I := 24 DOWNT0 1 DO
	PSHS A		
PNT611	LSR ,U	*	LSR [U]:MAN
	ROR 1,U		
	ROR 2,U		
	ROR 6,U	*	ROR RESULT
	ROR 7,U		
	ROR 8,U		
	LDA 6,U	*	CORRECT
	TFR A,B		
	CMPA #S80		
	BLO PNT612		
	SUBB #S30		
PNT612	TFR B,A		
	ANDA #S0F		
	CMPA #S08		
	BLO PNT613		
	SUBB #S03		
PNT613	STB 6,U		
	LDA 7,U		
	TFR A,B		
	CMPA #S80		
	BLO PNT614		
	SUBB #S30		
PNT614	TFR B,A		
	ANDA #S0F		
	CMPA #S08		
	BLO PNT615		
	SUBB #S03		
PNT615	STB 7,U		
	LDA 8,U		
	TFR A,B		
	CMPA #S80		
	BLO PNT616		
	SUBB #S30		
PNT616	TFR B,A		
	ANDA #S0F		
	CMPA #S08		
	BLO PNT617		
	SUBB #S03		
PNT617	STB 8,U		
	DEC ,S		
	BNE PNT611		
	LEAS 1,S		
	LEAU 3,U		
	RTS		

```

*****
*
*   SUBROUTINE INCH READS THE STATUS OF THE COLUMN AND ROW LINES OF
*   THE KEYPAD AFTER A KEY HAS BEEN DEPRESSED AND DECODES THESE LINES
*   TO GIVE A CHARACTER.
*
*   ENTRY:  NONE
*
*   EXIT:  ACCA CONTAINS THE CHARACTER
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 2 BYTES
*
*****
INCH  LDA KEYCR      * REPEAT
      BPL INCH       *   READ(KEYCR)
                          * UNTIL DATA READY
      LDA KEYBRD     * READ(KEYBRD)
      BPL PNT050     * IF B7 <> 0
                          *   THEN
      LSLA           *   IF B6 <> 0
      BPL PNT051
                          *   THEN
      LSLA           *   IF B5 <> 0
      BPL PNT052
                          *   THEN
      LSLA           *   IF B4 <> 0
                          *   THEN
      BMI INCH       *   GO TO INCH
                          *   ELSE
      LDB #50C       *   B := 12
      LSLA
      BRA PNT053
                          *   ELSE
PNT052 LDB #508      *   B := 8
      LSLA
      LSLA
      BRA PNT053
                          *   ELSE
PNT051 LDB #504      *   B := 4
      LSLA
      LSLA
      LSLA
      BRA PNT053
                          *   ELSE
PNT050 LDB #500      *   B := 0
      LSLA
      LSLA
      LSLA
      LSLA
PNT053 BPL PNT054    * IF B3 <> 0
                          *   THEN
                          *   IF B2 <> 0
                          *   IF B1 <> 0
                          *   IF B0 <> 0

```

```

BPL PNT055
LSLA
BPL PNT056
LSLA
BMI INCH
INCB
PNT056 INCB
PNT055 INCB
PNT054 LDX #KEYTAB
LDA B,X
RTS

KEYTAB FCC /CYNR/
FCC /7410/
FCC /852./
FCC /9631/

```

```

*      THEN
*      IF B1 <> 0
*
*      THEN
*      IF B0 <> 0
*      THEN
*      GO TO INCH
*      ELSE
*      B := B + 3
*      ELSE
*      B := B + 2
*      ELSE
*      B := B + 1
* X := KEYTAB
* A := [B+X]

```



```

*****
*
*   SUBROUTINE RDANSW INPUTS A YES OR NO ANSWER FROM THE KEYPAD.
*
*   ENTRY:   NONE
*
*   EXIT:   ACCA CONTAINS THE CHARACTER 'Y' OR 'N'
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 8 BYTES
*
*****
RDANSW LEAS -1,S      * RESERVE TEMP
PNT05A JSR INCH        * INCH
                   * CASE A OF
                   CMPA #'Y
                   BEQ PNT05B
                   CMPA #'N
                   BEQ PNT05B
                   CMPA #'I      * ENTER:
                   BNE PNT05C
                   LDA ,S+      *   A := TEMP
                   ANDCC #$FD    *   V := 0
                   RTS
PNT05B PSHS A          * Y,N:
                   JSR CRLF      *   CRLF
                   PULS A
                   JSR OUTCH     *   OUTCH
                   STA ,S        *   TEMP := A
                   BRA PNT05A    *   GO TO PNT05A
PNT05C CMPA #'R        *   0,...,9,DP,C:
                   BNE PNT05A    *   GO TO PNT05A
                   LEAS 1,S      * REPEAT: REMOVE TEMP
                   ORCC #$02     *   V := 1
                   RTS

```

```

*****
*
*   SUBROUTINE RDKEY INPUTS A REAL NUMBER FROM THE KEYPAD AND DISPLAYS
*   THIS NUMBER AS IT IS BEING KEYED IN.
*
*   ENTRY:  NONE
*
*   EXIT:   REAL NUMBER IS PUSHED ONTO THE TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S =  0 BYTES
*                  U = 18 BYTES
*
*****

RDKEY  CLR ,-S          * DISFLG := FALSE
      CLR ,-S          * FRCFLG := FALSE
      LDD #00000        * NUM := 0
      PSHS D
      PSHS D
      PSHS D            * DIV := 1
      LDD #040C0
      PSHS D
PNT05D JSR INCH         * INCH
      LDB 9,S          * IF NOT DISFLG
      BNE PNT05E
      * THEN
      PSHS A
      JSR CRLF         * CRLF
      PULS A
      DEC 9,S          * DISFLG := TRUE
PNT05E CMPL #09         * CASE A OF
      BHI PNT05F
      JSR OUTCH        * 0,...,9: OUTCH
      PSHS A           * NUM := NUM * 10 + A
      LDD 7,S
      PSHU D
      LDD 5,S
      PSHU D
      LDD #00000
      PSHU D
      LDD #04250
      PSHU D
      JSR MLTPLY
      LDA ,S+
      BEQ PNT060
      PSHS A
      LDD #00000
      PSHU D
      PULS B
      LDA #088
PNT061 DECA
      LSLB
      BPL PNT061

```

```

LSRA
RORB
PSHU D
JSR PLUS
PNT060 PULU D
STD 4,S
PULU D
STD 6,S
LDA 8,S
* IF FRCFLG = FALSE
* THEN
* GO TO PNT05D
* ELSE
* DIV := DIV * 10

LDD 2,S
PSHU D
LDD ,S
PSHU D
LDD #$0000
PSHU D
LDD #$4250
PSHU D
JSR MLTPLY
PULU D
STD ,S
PULU D
STD 2,S
BRA PNT05D
* GO TO PNT05D
PNT05F CMPA #' .
BNE PNT062
JSR OUTCH
* DP: OLICH
* FRCFLG := TRUE
LDA #$FF
STA 8,S
BRA PNT05D
* GO TO PNT05D
PNT062 CMPA #' |
BNE PNT063
LDD 6,S
* ENTER: [U] := NUM / DIV
PSHU D
LDD 4,S
PSHU D
LDD 2,S
PSHU D
LDD ,S
PSHU D
JSR INVRSE
JSR MLTPLY
LEAS 10,S
* REMOVE NUM, DIV, FRCFLG, DISFLG
ANDCC #$2D
* V := 0
RTS
PNT063 CMPA #' C
BNE PNT064
LEAS 10,S
* CLEAR: REMOVE NUM, DIV, FRCFLG, DISFLG
JSR CRLF
* CRLF
BRA RDKEY
* RDKEY
PNT064 CMPA #' R
BNE PNT05D
* OTHER: GO TO PNT05D

```


LEAS 10,S
ORCC #S02
RTS

* REPEAT: REMOVE NUM, DIV, FRCFLC, DISFLC
* V := 1

```

*****
*
*   SUBROUTINE SETPWR SETS THE OUTPUT VOLTAGE OF THE HP6130C DIGITAL
*   VOLTAGE SOURCE TO THE VALUE SPECIFIED BY THE REAL NUMBER LOCATED ON
*   THE TOP OF THE USER STACK. CHECKS ARE MADE TO INSURE THAT THE VOLTAGE
*   SET IS NOT LESS THAN ZERO VOLTS NOR GREATER THAN THE VOLTAGE SPECIFIED
*   BY THE REAL NUMBER LOCATED AT PSMAX. THE ACTUAL VALUE SET IS STORED
*   AT UO.
*
*   ENTRY:  DESIRED VOLTAGE SETTING CONTAINED IN TOP 4 BYTES OF USER STACK
*           MAXIMUM VOLTAGE SETTING CONTAINED IN 4 BYTES AT PSMAX
*
*   EXIT:   HP6130C UPDATED
*           ACTUAL VOLTAGE SETTING STORED AT UO
*
*   VOLATILE REGISTERS:  A, B, X, Y, CC
*
*   STACK USAGE:  S = 6 BYTES
*                 U = 14 BYTES
*
*****

```

```

*****
SETPWR LDD #S0000      * IF X <= 0.25
      PSRU D
      LDD #S3FC0
      PSRU D
      LEAX 4,U
      LEAY ,U
      JSR CNPXY
      BHI PNT065

      * THEN
      LEAU 8,U          * REMOVE X, 0.25
      LDD #S0000        *   UO := 0
      STD UO
      STD UO+2
      LDA PSFLGS        *   X10 := 0
      ANDA #S7F
      STA PSFLGS

      * REPEAT
PNT066 LDA PSFLGS      *   READ(PSFLGS)
      BPL PNT066        *   UNTIL READY TO SEND
      LDD #S0000        *   PSDATA := 0
      STD PSDATO
      RTS

      * ELSE
PNT065 LDD PSMAX+2     *   IF X >= PSMAX
      STD 2,U
      LDD PSMAX
      STD ,U
      JSR CNPXY
      BLO PNT067

      * THEN
      LDD 2,U          *   X := PSMAX
      STD 6,U
      LDD ,U

```

PNT067	STD 4,U	* REMOVE PS:MAX
	LEAU 4,U	* UO := X
	LDD ,U	
	STD UO	
	LDD 2,U	
	STD UO+2	
	LDD #\$FF80	* IF X >= 16383.75
	PSHU D	
	LDD #\$477F	
	PSHU D	
	JSR CMPXY	
	BLO PNT068	
		* THEN
	LDD #\$6666	* X := X / 10
	STD 2,U	
	LDD #\$3EE6	
	STD ,U	
	JSR MLTPLY	
	LDA PSFLGS	* X10 := 1
	ORA #\$03	
	STA PSFLGS	
	BRA PNT069	
		* ELSE
PNT068	LEAU 4,U	* REMOVE 16383.75
	LDA PSFLGS	* X10 := 0
	ANDA #\$F7	
	STA PSFLGS	
PNT069	LSL 3,U	* EXPAND X
	ROL 2,U	
	ROL 1,U	
	ROL ,U	
	LDA #\$8F	* FOR I := (\$8F - X-EXP) DOWNT0 1 DO
	SUBA ,U	
	PSHS A	
PNT06A	LSR 1,U	* SHIFT X-MAN RIGHT
	ROR 2,U	
	DEC ,S	
	BNE PNT06A	
	LEAS 1,S	
	BCC PNT06B	* ROUND X-MAN
	INC 2,U	
	BNE PNT06B	
	INC 1,U	
		* REPEAT
PNT06B	LDA PSFLGS	* READ(PSFLGS)
	BPL PNT06B	* UNTIL READY
	LDD 1,U	* PS:DATA := X-MAN
	STD PS:DATA	
	LEAU 4,U	* REMOVE X
	RTS	

```

*****
*
*   SUBROUTINE SETCUR PROMPTS THE USER FOR THE DESIRED OUTPUT CURRENT
*   LIMIT FOR THE HP6130C DIGITAL VOLTAGE SOURCE. THIS ROUTINE THEN
*   READS THE DESIRED LIMIT FROM THE KEYPAD AND COMPARES THE DESIRED LIMIT
*   WITH THE 8 POSSIBLE VALUES OF THE CURRENT LIMIT. THIS ROUTINE THEN
*   SETS THE CURRENT LIMIT TO THE CLOSEST MATCH TO THE DESIRED VALUE AND
*   RETURNS THE SET VALUE OF THE CURRENT LIMIT ON THE TOP OF THE USER STACK.
*   THE NEW CURRENT LIMIT VALUE IS SENT TO THE HP6130C ON THE NEXT CALL TO
*   SUBROUTINE SETPWR.
*
*   ENTRY:  NONE
*
*   EXIT:  NEW CURRENT LIMIT SETTING CONTAINED IN TOP 4 BYTES OF THE USER
*          STACK
*
*   VOLATILE REGISTERS:  A, B, X, Y, CC
*
*   STACK USAGE:  S = 11 BYTES
*                  U = 4 BYTES
*
*****

```

```

*****
SETCUR JSR CRLF      * REPEAT
        JSR SCRSTR   *   CRLF
        FCC /WHAT /  *   WRITE('WHAT IS THE POWER SUPPLY OUTPUT CURRENT
        FCC /IS TH/  *   LIMIT IN MILLIAMPS?')
        FCC /E POW/
        FCC /ER SU/
        FCC /PPLY /
        FCC /OUTPU/
        FCC /T CUR/
        FCC /RENT /
        FCC /LIMIT/
        FCC / IN M/
        FCC /ILLIA/
        FCC /MPS?/
        FCB EOT
        JSR RDKEY    *   RDKEY
        BVS SETCUR   *   UNTIL V CLEAR
        LDD #0000     * IF X <= 35
        PSHU D
        LDD #4346
        PSHU D
        LEAX 4,U
        LEAY ,U
        JSR CMPXY
        BHI PNT06C

        *   THEN
        *   CURRENT := 20

        LDA PSFLGS
        ORA #07
        STA PSFLGS
        LEAU 8,U      *   REMOVE X, 35
        LDD #0000     *   [U] := 20
        PSHU D
*****

```

LDD #S42D0		
PSHU D		
BRA PT0073		
PNT06C LDD #S0000	*	ELSE
STD 2,U	*	IF X <= 60
LDD #S4378		
STD ,U		
JSR CMPXY		
BHI PNT06D		
LDA PSFLGS	*	THEN
ANDA #SF8	*	CURRENT := 50
ORA #S06		
STA PSFLGS		
LEAU 8,U	*	REMOVE X, 60
LDD #S0000	*	[U] := 50
PSHU D		
LDD #S4364		
PSHU D		
BRA PT0073		
PNT06D LDD #S0000	*	ELSE
STD 2,U	*	IF X <= 85
LDD #S43D5	.	
STD ,U		
JSR CMPXY		
BHI PNT06E		
LDA PSFLGS	*	THEN
ANDA #SF8	*	CURRENT := 70
ORA #S05		
STA PSFLGS		
LEAU 8,U	*	REMOVE X, 85
LDD #S0000	*	[U] := 70
PSHU D		
LDD #S43C6		
PSHU D		
BRA PT0073		
PNT06E LDD #S0000	*	ELSE
STD 2,U	*	IF X <= 150
LDD #S444B		
STD ,U		
JSR CMPXY		
BHI PNT06F		
LDA PSFLGS	*	THEN
ANDA #SF8	*	CURRENT := 100
ORA #S04		
STA PSFLGS		
LEAU 8,U	*	REMOVE X, 150
LDD #S0000	*	[U] := 100
PSHU D		

LDD #S43E4		
PSHU D		
BRA PT0073		
PNT06F LDD #S8000	*	ELSE
STD 2,U	*	IF X <= 350
LDD #S44D7		
STD ,U		
JSR CMPXY		
BHII PNT070		
LDA PSFLGS	*	THEN
ANDA #SF8	*	CURRENT := 200
ORA #S03		
STA PSFLGS		
LEAU 8,U	*	REMOVE X, 350
LDD #S0000	*	[U] := 200
PSHU D		
LDD #S4464		
PSHU D		
BRA PT0073		
PNT070 LDD #S0000	*	ELSE
STD 2,U	*	IF X <= 600
LDD #S454B	.	
STD ,U		
JSR CMPXY		
BHII PNT071		
LDA PSFLGS	*	THEN
ANDA #SF8	*	CURRENT := 500
ORA #S02		
STA PSFLGS		
LEAU 8,U	*	REMOVE X, 600
LDD #S0000	*	[U] := 500
PSHU D	,	
LDD #S44FD		
PSHU D		
BRA PT0073		
PNT071 LDD #S4000	*	ELSE
STD 2,U	*	IF X <= 850
LDD #S456A		
STD ,U		
JSR CMPXY		
BHII PNT072		
LDA PSFLGS	*	THEN
ANDA #SF8	*	CURRENT := 700
ORA #S01		
STA PSFLGS		
LEAU 8,U	*	REMOVE X, 850
LDD #S8000	*	[U] := 700
PSHU D		

LDD #S4557		
PSHU D		
BRA PT0073	*	ELSE
	*	CURRENT := 1000
PNT072 LDA PSFLGS		
ANDA #SF8		
STA PSFLGS		
LEAU 8,U	*	REMOVE X, 850
LDD #S0000	*	[U] := 1000
PSHU D		
LDD #S457D		
PSHU D		
PT0073 LDD 2,U	* PSCURR := [U]	
STD PSCURR+2		
LDD ,U		
STD PSCURR		
RTS		

```

*****
*
*   SUBROUTINE RDPWR READS THE PRESENT VOLTAGE SETTING OF THE HP6130C
*   DIGITAL VOLTAGE SOURCE AND RETURNS THIS VALUE ON THE TOP OF THE USER
*   STACK.
*
*   ENTRY:  NONE
*
*   EXIT:  VOLTAGE SETTING CONTAINED IN TOP 4 BYTES OF USER STACK
*
*   VOLATILE REGISTERS:  A, B, X, CC
*
*   STACK USAGE:  S = 5 BYTES
*                  U = 18 BYTES
*
*****
RDPWR  LDD PSDATO      * D := PSSET
      BNE PNT073      * IF PSSET = 0
                      * THEN
                      *   GO TO ZERO
PNT073 LEAU -8,U      * RESERVE MEMORY
      LSLB            * X-MAN := ABS(PSSET)
      ROLA
      STD 2,U
      BCC PNT074      * IF PSSET < 0
                      * THEN
                      *   X-SGN := NEGATIVE
      LDA #SFF        *
      STA ,U
      BRA PNT075      * ELSE
                      *   X-SGN := POSITIVE
PNT074 CLR ,U
PNT075 LDD #S0000
      STD 4,U
      STD 6,U
      LDA #S8E        * X-EXP := 14
      STA 1,U
      JSR LFTJST      * LEFT JUSTIFY X AND SHRINK
      LDA PSFLGS      * IF X10 = 1
      ANDA #S03
      BEQ PNT076
                      * THEN
                      *   X := X * 10
      LDD #S0000
      PSHU D
      LDD #S4250
      PSHU D
      JSR MLTPLY
PNT076 RTS

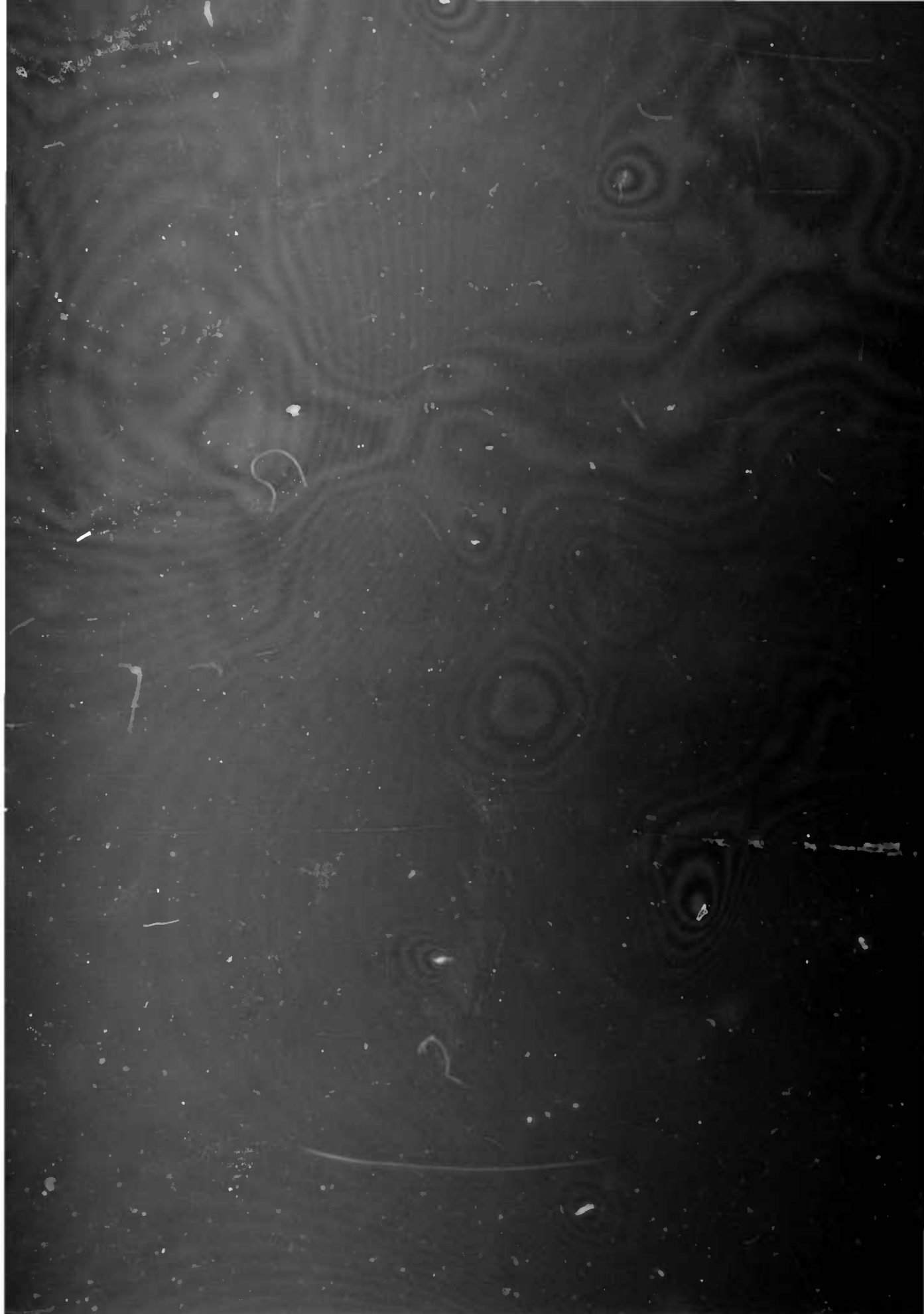
```


AD-A118 040 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 13/1
AN AUTOMATED TEMPERATURE CONTROLLER FOR THE ADVANCED HALL EFFEC--ETC(11)
MAR 82 D J PAGE
UNCLASSIFIED AFIT/GEO/EE/82M-1

NL

3 of 3
AD: A
118040

END
DATE
FILMED
9 82
DTIC



D- A
804

```

*****
*
*   SUBROUTINE PWRSEN APPLIES BIASING VOLTAGE TO EACH SECONDARY SENSOR
*   WHOSE SENSOR DATA INDICATES THAT INTERNAL BIASING IS REQUIRED BASED
*   UPON THE PRESENT VALUE OF THE REAL NUMBER LOCATED AT YD.
*
*   ENTRY:  SENSOR DATA FOR ALL SENSORS COMPLETED
*           4 BYTES AT YD MUST CONTAIN NEXT DESIRED SENSOR INPUT SETTING
*
*   EXIT:  APPROPRIATE SENSORS ARE BIASED WITH A CONSTANT VOLTAGE
*
*   VOLATILE REGISTERS:  A, B, X, Y, CC
*
*   STACK USAGE:  S = 6 BYTES
*
*****
PWRSEN LDA #S01      * FOR I := 1 TO NUMSEN DO
      PSHS A
PNT030 LDA ,S
      CMPA NUMSEN
      BHI PNT077
      LDY #SEN1      *   Y := SEN(I)
      LDA ,S
      DECA
      LDB #$11
      MUL
      LEAY B,Y
      LDA ,Y      *   IF SEN(I) = SECONDARY AND PWRRQD
      ANDA #$30
      CMPA #$30
      BNE PNT078
      *           THEN
      *           IF YD > SEN(I).PRNG1
      LEAY 9,Y
      LDX #YD
      JSR CMPXY
      BLS PNT079
      *           THEN
      *           IF YD <= SEN(I).PRNG2
      LEAY 4,Y
      JSR CMPXY
      BHI PNT07A
      *           THEN
      *           SENPWR(I) := ON
      LDA #$10
      LDB ,S
PNT07B ASRA
      DECB
      BNE PNT07B
      ORA SENFLG
      STA SENFLG
      BRA PNT078
      *           ELSE
      *           SENPWR(I) := OFF
PNT07A LDA #$EF
      LDB ,S
PNT07C ASRA
      DECB

```

BNE PNT07C	
ANDA SENFLG	
STA SENFLG	
BRA PNT078	
	*
PNT079 LEAY 4,Y	*
JSR CMPXY	
BLO PNT07D	
	*
LDA #S10	*
LDB ,S	
PNT07E ASRA	
DECB	
BNE PNT07E	
ORA SENFLG	
STA SENFLG	
BRA PNT078	
	*
PNT07D LDA #SEF	*
LDB ,S	
PNT07F ASRA	
DECB	
BNE PNT07F	
ANDA SENFLG	
STA SNEFLG	
PNT078 INC ,S	
BRA PNT080	
PNT077 LEAS 1,S	
RTS	

ELSE
IF YD >= SEN(I).PRNG2

THEN
SENPWR(I) := ON

ELSE
SENPWR(I) := OFF

```

*****
*
*   SUBROUTINE RDSN READS THE VALUE OF THE A/D DATA LINES, THE A/D
*   STATUS LINES, AND THE PROGRAMMED AMPLIFIER GAIN AND RETURNS THE INPUT
*   VALUE AS A REAL NUMBER ON THE TOP OF THE USER STACK.  THIS ROUTINE
*   ALSO CHECKS FOR INVALID A/D DATA AND IF SO GENERATES A SOFTWARE
*   INTERRUPT.  THE PROGRAMMED AMPLIFIER GAIN IS ALSO ADJUSTED TO GIVE THE
*   MAXIMUM INPUT SIGNAL TO THE A/D CONVERTER WITHOUT GENERATING AN
*   OVERLOAD.
*
*   ENTRY:  A/D IS RUNNING
*           A/D INPUT IS ENABLED
*           ONE SENSOR IS SELECTED
*
*   EXIT:   A/D INPUT IS CONTAINED IN TOP 4 BYTES OF USER STACK
*           A/D IS HELD
*
*   VOLATILE REGISTERS:  A, B, CC
*
*   STACK USAGE:  S = 16 BYTES
*                 U = 8 BYTES
*
*****

```

```

RDSN:  LDA SENC2      * REPEAT
      BPL RDSN        *   READ(SENCR2)
                        * UNTIL DATA READY
      LDD SENDAO      * READ(SENDAO)
      PSHS D
      LDD SENFLG      * READ(SENFLG)
      PSHS D
      ANDB #SBF        * HOLD A/D
      ORB #S80        * DISABLE INPUT
      STB SENSEL
      ANDA #S20        * IF A/D STATUS = INVALID
      BEQ PNT081
                        * THEN
                        *   SWI(3)
      SWI
      FCB S03
      PSHS D
      LDD SENDAO      *   CLEAR DATA MISSED FLAG
      PULS D
PNT081 LDA ,S          * IF A/D OVERRUN
      ANDA #S40
      BEQ PNT082
                        * THEN
                        *   IF GAIN <> 1
      LDA 1,S
      ANDA #S03
      BEQ PNT083
                        * THEN
                        *   GAIN := GAIN / 4
      DEC SENSEL
      BRA PNT084
                        * ELSE
                        *   SWI(4)
PNT083 SWI
      FCB S04

```

LDD #FFFF	*	DATA := 7.999878
STD 2,S		
BRA PNT084		
PNT082 LDD 2,S	*	ELSE
CMPD #SE000	*	IF DATA > 7
BLS PNT085		
LDA 1,S	*	THEN
ANDA #S03	*	IF GAIN > 1
BEQ PNT084		
DEC SENSEL	*	THEN
BRA PNT084	*	GAIN := GAIN / 4
PNT085 CMPD #S2000	*	ELSE
BHS PNT084	*	IF DATA < 1
LDA 1,S	*	THEN
ANDA #S03	*	IF GAIN < 64
CMPA #S03		
BEQ PNT084		
INC SENSEL	*	THEN
PNT084 LDD #S0000	*	GAIN := GAIN * 4
PSHU D	*	FORM X-MAN
PSHU D		
LDD 2,S		
PSHU D		
LDB #S23	*	FORM X-EXP
LDA 1,S		
ANDA #S03		
BEQ PNT086		
PNT087 SUBB #S02		
DECA		
BNE PNT087		
PNT086 PSHU B		
LDB ,S	*	FORM X-SGN
SEX		
COM A		
PSHU A		
LEAS 4,S	*	REMOVE SENFLG, DATA
JSR LFTJST	*	LEFT JUSTIFY X AND SHRINK
LDA SENSEL	*	ENABLE INPUT
ANDA #S7F		
STA SENSEL		
RTS		

```

*****
*
*   DEFINED BELOW ARE THE VECTOR LOCATIONS FOR THE MC6809 MICROPROCESSOR.
*   ALL INTERRUPTS AND RESETS CAUSE THE MICROPROCESSOR TO AUTOMATICALLY
*   ACCESS THESE LOCATIONS TO DETERMINE THE BEGINNING LOCATION OF EACH OF
*   THE RESPECTIVE ROUTINES.
*
*****
      ORG VECTOR
VSW13 FDB SWIVEC
VSW12 FDB SWIVEC
VFIRQ FDB FIRQ
VIRQ  FDB IRQVEC
VSWI  FDB SWIVEC
VNMI  FDB NNIVEC
VRESET FDB RSTVEC
      END

```


Appendix C
Operator's Manual

Contained in this appendix is a short description of the external connections, required information, panel switches, display and error messages, and peripheral interfaces of the automatic temperature controller. More complete information on each of these areas can be obtained from Dr. Patrick M. Hemenger, AFWAL/MLPO, Wright-Patterson AFB, Ohio 45433.

External Connections

All connections between the automatic temperature controller and the thermometers, the HP 6130C Digital Voltage Source, and the computer are made through standard RS-232 female connectors. These connectors are located on the back panel of the controller.

A temperature controller can handle the input signals from up to four thermometers. These thermometers can produce either current or voltage signals. A 10 millivolt constant voltage source is provided for each thermometer. This source may be used to bias current signal thermometers which are not biased externally.

Each thermometer must be connected to the controller beginning with the -1 pins, then the -2 pins, etc. Voltage signal thermometers are connected between the VOLT and GND pins. Externally biased thermometers should be connected between the CURRENT and GND pins. Internally biased thermometers should be connected between the CURRENT and SOURCE pins.

Pin assignments for the thermometer connector are as follows:

<u>Pin</u>	<u>Pin-Outs</u>	<u>Pin</u>	<u>Pin-Outs</u>
1	VOLT-1	7	VOLT-3
2	CURRENT-1	8	CURRENT-3
14	GND-1	20	GND-3
15	SOURCE-1	21	SOURCE-3
4	VOLT-2	10	VOLT-4
5	CURRENT-2	11	CURRENT-4
17	GND-2	23	GND-4
18	SOURCE-2	24	SOURCE-4

The HP 6130C Digital Voltage Source is connected to the automatic temperature controller with a cable between the J1 connector, located on the back of the HP 6130C, and the RS-232 connector of the controller. The pin assignments for the RS-232 connector to the HP 6130C are as follows:

<u>Pin</u>	<u>Pin-Outs</u>	<u>Pin</u>	<u>Pin-Outs</u>
1	GATE	14	FLAG
2	V SIGN	15	D0
3	D1	16	D2
4	D3	17	LATCH STAT
5	D4	18	OVLD STAT
6	D5	19	D6
7	D7	20	D8
8	D9	21	V RANGE
9	D10	22	D11
10	D12	23	L22
11	D13	24	L23
12	D14	25	L24
13	GND		

The internal connections between the microcomputer board and the RS-232 connector designated for the computer were not completed during the implementation of the controller. Therefore, the user must make the connections between H-13 of the microprocessor board and the RS-232 connector. Once these connections are made, a patch-cord between the RS-232 connector of the controller and the J1 connector of the DRV-11

parallel interface card on the LSI-11 computer will complete the hardware interface.

Required Information

Once the automatic temperature controller is powered up, it will begin to prompt the user for required information. The following is a list of the prompts and an explanation of what each means:

HOW MANY SENSORS WILL BE USED(1,2,3,4)?

How many thermometers have been connected to the automatic temperature controller?

IS SENSOR 1 THE PRIMARY SENSOR(Y,N)?

Is the 1th thermometer to be monitored when changing from one temperature to another? Answering NO causes the controller to use the 1th thermometer as a control thermometer.

DOES SENSOR 1 HAVE A POSITIVE SLOPE(Y,N)?

Does the 1th thermometer produce a signal which is monotonically increasing with temperature?

DOES SENSOR 1 PRODUCE A VOLTAGE SIGNAL(Y,N)?

Is the signal produced by the 1th thermometer a voltage signal? Answering NO causes the controller to assume that the 1th thermometer produces a current signal.

DOES SENSOR 1 REQUIRE INTERNAL POWER(Y,N)?

Does the 1th thermometer require the use of the 10 millivolt internal biasing source?

OVER WHAT RANGE SHOULD SENSOR 1 BE USED START POINT? END POINT?

What temperature range should the 1th thermometer, a control thermometer, be used to control temperature? The temperature range must be expressed as the corresponding input signal levels of the primary thermometer, the thermometer used to adjust temperature.

OVER WHAT RANGE SHOULD SENSOR 1 BE POWERED? START POINT? END POINT?

What temperature range should the constant voltage biasing source be applied to the 1th thermometer? The temperature range must be expressed as the corresponding input signal levels of the primary thermometer.

WHAT IS THE DESIRED STEADY-STATE ERROR?

How close does the measured input signal have to be to the desired input signal level of the primary thermometer before the controller is allowed to switch to one of the control thermometers?

WHAT IS THE POWER SUPPLY OUTPUT CURRENT LIMIT IN MILLIAMPS?

What should the output current through the heater coil be limited to?

WHAT IS THE LOAD RESISTANCE IN OHMS?

What is the resistance of the heater coil?

Panel Switches

Certain controller operations can be controlled using the panel switches located on the front of the automatic temperature controller. These switches include a toggle switch, a rotary switch, a reset switch, and a power switch.

The toggle switch indicates to the temperature controller whether the desired input signal level will be supplied by the computer or the user. With the switch in the COMPUTER position, temperature adjustment and control is done automatically. With the switch in the MANUAL position, the user must enter the desired input signal level for each new temperature set point.

The rotary switch determines which thermometer will be used for temperature control. Five positions are available on this switch: AUTO, 1, 2, 3, and 4. With the switch in the AUTO position, the controller will determine which thermometer should be used for control based upon the desired input signal level and the information provided by the user on each of the control thermometers. With the switch in any other position, the corresponding thermometer will be used for con-

trolling temperature. If the thermometer selected does not exist, the highest numbered thermometer will be used for temperature control. For example, if the switch is in position 4 and only 2 thermometers exist, then thermometer 2 will be used for control.

The two remaining switches, the power switch and the reset switch, are used for applying power to the temperature controller and resetting the microprocessor system. The microprocessor system is automatically reset upon power-up. If the reset switch is closed, all information entered by the user or the computer is lost and the controller will begin prompting the user to enter this information again.

Display and Error Messages

During the normal operation of the automatic temperature controller, the display is used for prompting information from the user and indicating the current status of the controller. The normal display during temperature adjustment and temperature control is:

ND +C.XXXXXX/+XX

where

N is the number of the present thermometer being monitored

D is the direction of change from the last thermometer sample

+0.XXXXXX/+XX is the present thermometer reading in volts for thermometers which produce voltage signals and hundreds of microamperes for thermometers which produce current signals.

In addition to the normal display format, the following error messages are also included along with an explanation of each:

INVALID INPUT

The data which have just been entered are invalid or are inconsistent with the data which have been previously entered.

SENSOR 1 MUST BE POWERED OVER THE ENTIRE USABLE RANGE

The ith thermometer has been determined to require the use of the internal voltage biasing source. The range over which the voltage biasing source is to be used, however, does not provide a biasing source for the ith thermometer over the entire range that the ith thermometer is to be used.

PRIMARY SENSOR NOT SPECIFIED . . . REENTER DATA

None of the thermometers specified has been identified as the thermometer to be used for adjusting the temperature.

SECONDARY SENSOR UNDETERMINED . . . WHICH SENSOR SHOULD BE USED?

The thermometer to be used for temperature control cannot be determined from the information provided on each of the thermometers. Which one of the available thermometers should be used for temperature control?

POWER SUPPLY CURRENT LIMIT EXCEEDED. SHOULD THIS LIMIT BE INCREASED?

The HP 6130C Digital Voltage Source has determined that the specified maximum heater coil current has been exceeded. Should the maximum heater coil current limit be increased?

HARDWARE FAILURE . . . DEFAULT?

The input data from the toggle switch is invalid and it is suspected that the problem resides in a broken connection in the switch hardware. Should the controller assume that input is to be entered from the keypad? Answering NO will terminate the controller program.

UNDEFINED COMPUTER INPUT . . . DEFAULT?

The desired input signal level provided by the computer is an undefined number. Should the controller assume that this number is zero? Answering NO will terminate the controller program.

DIVISION BY ZERO . . . DEFAULT?

An attempt has been made to perform division by zero. Should the controller assume that the result of this division is a large positive number? Answering NO will terminate the controller program.

A/D DATA MISSED . . . DEFAULT?

The A/D status line was high when the A/D converter was

read indicating that the data may not be valid. Should this error be ignored? Answering NO will terminate the controller program.

A/D OVERLOAD . . . DEFAULT?

The signal being measured by the A/D converter is too large for the A/D to handle even with minimum gain from the programmable gain stage. Should the controller assume that this is a full scale reading and continue execution? Answering NO will terminate the controller program.

Peripheral Interfaces

This section contains a functional description of each bit of the PIA peripheral data and control registers for each of the peripheral interfaces: The display interface, the switch and keypad interface, the heater coil interface, the thermometer interface, and the computer interface.

DISPLAY INTERFACE

ADDR 1000 b3b2b1b0	Digit position of where the next character is to be placed on the display.
ADDR 1001 b7b6b5b4b3b2b1b0	Character to be output to the display.
ADDR 1010 b5b4b3	Controls the display enable.
ADDR 1011 b5b4b3	Controls the output strobe pulse for ADDR 1001.

SWITCH AND KEYPAD INTERFACE

ADDR 1002 b6b5	Input data from the toggle switch.
ADDR 1002 b4b3b2b1b0	Input data from the rotary switch.
ADDR 1003 b7b6b5b4b3b2b1b0	Input data from the keyboard.
ADDR 1012 b7b6	Switch change-of-state detection bits.
ADDR 1012 b5b4b3	Switch edge detection and interrupt control bits.

ADDR 1012 b1b0	Switch edge detection and interrupt control bits.
ADDR 1013 b7	Keypad key detection bit.
ADDR 1013 b1b0	Keypad edge detection and interrupt control bits.

HEATER COIL INTERFACE

ADDR 1005 b7	Busy flag from the HP 6130C.
ADDR 1005 b6	Current Latch Status from HP 6130C.
ADDR 1005 b3	Voltage Range output to HP 6130C.
ADDR 1005 b2b1b0	Current Limit output to HP 6130C.
ADDR 1006 b7	Voltage Sign output to HP 6130C.
ADDR 1006 b6b5b4b3b2b1b0	Most Significant Voltage Magnitude output to HP 6130C.
ADDR 1007 b7b6b5b4b3b2b1b0	Least Significant Voltage Magnitude output to HP 6130C.
ADDR 1015 b7	Voltage Source Overload detection bit.
ADDR 1015 b1b0	Voltage Source Overload edge detection and interrupt control bits.
ADDR 1017 b5b4b3	Controls the output strobe pulse for ADDR 1007.

THERMOMETER INTERFACE

ADDR 1008 b7	A/D input data polarity.
ADDR 1008 b6	A/D input data overrun indication bit.
ADDR 1008 b5	A/D status indication bit.
ADDR 1008 b3b2b1b0	Constant Voltage Source output control bits.
ADDR 1009 b7	A/D signal input control bit.
ADDR 1009 b6	A/D run/hold control bit.

ADDR 1009 b5b4b3b2	Thermometer select bits.
ADDR 1009 b1b0	Programmable gain control bits.
ADDR 100A b7b6b5b4b3b2b1b0	Most Significant A/D input data.
ADDR 100B b7b6b5b4b3b2b1b0	Least Significant A/D input data.
ADDR 1019 b5b4b3	Thermometer select enable bits.
ADDR 101A b7b6	A/D status change-of-state detection bits.
ADDR 101A b5b4b3	A/D status edge detection and interrupt control bits.
ADDR 101A b1b0	A/D status edge detection and interrupt control bits.

COMPUTER INTERFACE

ADDR 100C b7b6b5b4b3b2b1b0	Most Significant Computer input data.
ADDR 100D b7b6b5b4b3b2b1b0	Least Significant Computer input data.
ADDR 101C b7	Computer input data detection bit.
ADDR 101C b5b4b3b1b0	Handshake signal control bits.
ADDR 101D b5b4b3	Temperature-set output indication bit.

Vita

Daniel John Page was born 13 May 1957 in Sioux Falls, South Dakota. He graduated from Lincoln Senior High School, Sioux Falls, South Dakota, in 1975. He then attended South Dakota State University in Brookings, South Dakota, where he received the Bachelor of Science degree with a major in Electrical Engineering in 1979. He was commissioned at this time in the United States Air Force.

After graduation, he continued his studies at South Dakota State University until he reported for active duty. He reported to the School of Engineering, Air Force Institute of Technology, in May 1980.

He is married to the former Laura Elfriede Huguen.

Permanent Address: 809 East 33rd Street

Sioux Falls, South Dakota 57105

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GEO/EE/82M-1	2. GOVT ACCESSION NO. AD-A118040	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN AUTOMATED TEMPERATURE CONTROLLER FOR THE ADVANCED HALL EFFECT EXPERIMENTAL DATA ACQUISITION SYSTEM	5. TYPE OF REPORT & PERIOD COVERED M.S. Thesis	
7. AUTHOR(s) Daniel J. Page 2nd Lieutenant USAF	6. PERFORMING ORG. REPORT NUMBER	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433	8. CONTRACT OR GRANT NUMBER(s)	
11. CONTROLLING OFFICE NAME AND ADDRESS Lasers and Materials Branch (AFWAL/MLPO) Air Force Wright Aeronautical Laboratories/ML Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	12. REPORT DATE March 1982	
	13. NUMBER OF PAGES 206	
	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release; Distribution Unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE; LAW AFR 190-17 23 JUL 1982 LYNN E. WOLAVER Dean for Research and Professional Development AIR FORCE INSTITUTE OF TECHNOLOGY (ATC) WRIGHT-PATTERSON AFB, OH 45433		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Automation Hall Effect Temperature Control MC6809 Microprocessor Microprocessor Thermometer Temperature Measurement Model Following		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The Air Force Wright Aeronautical Laboratories, Materials Laboratory, conducts experiments using the Hall effect to characterize the electrical properties and impurity levels of silicon samples. An automated data acquisition system controls the conduct of the experiment and reduces all the necessary data. The purpose of this study was the development of an automated temperature controller to interface with the automated data acquisition system and the experiment. The temperature controller is designed to control the temperature of the		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED


SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. silicon sample to within 0.005 degrees Kelvin in the temperature range of 4.2 to 300 degrees Kelvin. The control algorithm measures the thermal impulse response of the system and uses this information to adjust and control the temperature.

An MC6809 microprocessor with 10K bytes of EPROM and 640 bytes of RAM is used to implement the controller. The control algorithm and other software was developed to enable the controller to control temperature. A number of problems with the present controller design are identified and recommendations for improvements to the design are made.



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

END

DATE
FILMED

9 - 82

DTIC